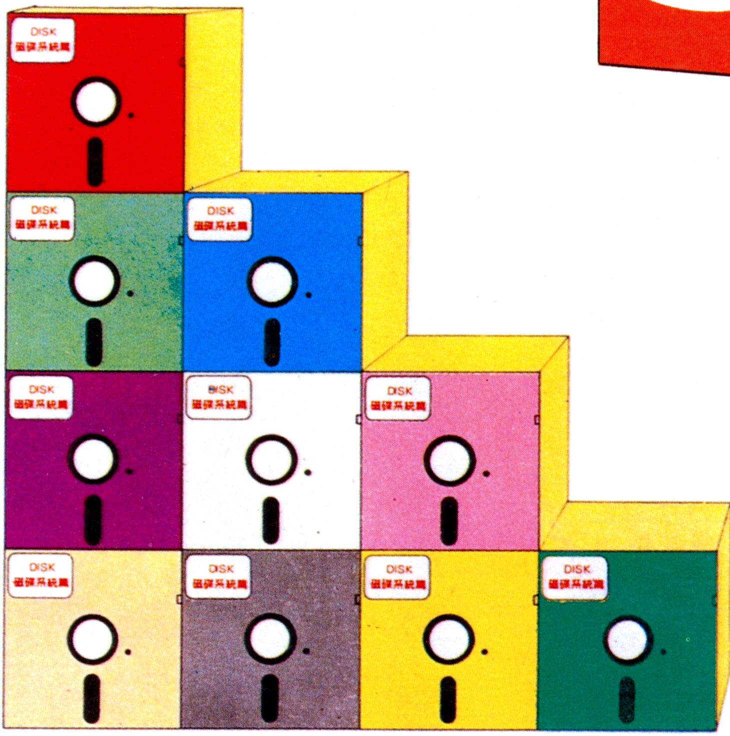
apple II





3 微電腦徹底研究

施威銘 著

APPLE
微電腦徹底研究

（三）

施 威 銘 著

APPLE Ⅱ微電腦徹底研究㈢

編　者：施威銘
出版者：資 料 出 版 社
地　址：香港洛克道347號
印刷者：者達印刷廠
地　址：香港柴灣工廠大厦23樓G座

唐凌　序

假如有一天，你用你自己的腦筋，寫了一本書，送到出版社去，或則只是費力地寫好一篇文章，投到某專業性雜誌去發表，你可能遭到一項極大的困難——出版商或雜誌的編輯會向你索取「原文」。

出版商或雜誌編輯之希望文有出處的心理，是不難理解的，一是沾光，一是如有謬誤可一推了之。

事實上，這樣的心態，普通地存在於現居台灣的每一個人心裡，而不只是出版界。

做事有個依據是對的，然而任何依據本身不可能是眞理，即或是眞理，在被引用的過程也可能造成扭曲。

下面就是一段隨處可見而扭曲得可笑的例子：

「在半計數（Counter halves）之間，外來的顫動（Jumper）必須供給。

「若經由Clock 1進入，而且Q 8跳躍至Clock 1，計數比重為1－2－4－5，Q 1是位元比重最大的輸出而且在輸出端有一個對稱的方波。……………

這是一段對 7490 動作的描述，如果你對 7490 毫無概念，那麼看了這一段，眞的會令你「顫動」不已。

不幸的事情是，對這樣荒唐的謬誤，出版商給予容忍，讀者也同樣給予容忍。於是成千上萬的學子們，只好一天天地「顫動」下去。

另外一種情況也是很可悲的，也就是不知從什麼時候開始，大學生都喜歡抱厚厚的洋裝書，使得一些類如三用表用法、OTL、OCL 設計或CMOS 玩具製作之類的小書，有興趣的人只好偸偸地看，總覺得這些小書，好像很不入流一樣。

我與施威銘先生只是初識，當我看完「 APPLE Ⅱ微電腦徹底研究」第一册之後，覺得與我主編的「音響技術月刊」頗有其臭相類之處，那就是作者躍然紙上，隨時與你共處。欲這樣，顯然是需要一些勇氣的，一些面對可能的疏失的勇氣，一些面對抱洋裝書的人的勇氣。我未嘗聽說有人照著食譜做菜，把菜燒焦了、燒爛了、糊了，會找作者理論一番的，但如果你寫技術的書、電腦的書，隨時你都得迎接焦了、爛了、糊了的挑戰。

總之，這不是一篇「序」，而是一點感想，我由衷地希望，每一位讀者都能珍惜作者施威銘先生在這本書中「掏」給你的東西。



序

徹底研究系列出版之後，很受到相當多數讀者的熱烈反應，這期間詢問最多的是：第 3 、 4 册何時出版？內容又是什麼？

就是這個問題“到底要以怎樣的方向擬定怎樣的內容？”，使得第三册的出版延時至今。這並不意味著不知如何來編寫這本書，而所考慮的是，既然有那樣多的讀者在熱烈的期望，那麼除了是“有趣的書”並希望以“愉快的心情來閱讀”（第一册序）之外、除了是各式各樣的 COPY 方法之外，我們是否還可以學習到更進一層的意義呢？

基於這樣的構想，以 APPLE 的 DOS 做爲主題，由理論的介紹（第 1 章）配合實務的解析（第 2 章），我們開始了一連串的 DOS 徹底研究。這其中包含了 I ／O ERROR 的處理（第 3 章）、DOS 的解剖、修改及應用（ 4 、 5 、 6 章），最後更介紹了以 RAM 來模擬磁碟機運作的方法（第 7 章）。所有內容由最低階的物理實體（第 0 章）到最高階的檔案觀念，均細加闡述，並脈絡串穿使之清楚的呈現。

我們的用意是，借由APPLE DOS 的解析來介紹電腦作業系統（ OS ）的概念，雖然APPLE DOS 與一般傳統OS尚有一段距離*，但只要徹底瞭解、觀念清楚，要進一步研習其他系統，是一點都不困難的。

因此，以愉快的心情、盎然的興緻，再用點心便可以了！

作者 施威銘

1984.02.20

---

*最近已有multitask、virtual storage的APPLE系統上市。

目錄

第0章　從頭開始：INIT

0－0　INIT？　1
0－1　磁片的物理結構　2
0－2　磁碟機的運轉：INIT的說明　5
0－3　位元組同步法　8
0－4　APPLE 磁軌的安排　10
0－5　最底層的 I／O　12
0－6　奇偶編碼法與 READADDR　23
0－7　下一步的工作？　28

第1章　檔案系統的理論

1－0　微電腦的檔案系統　38
1－1　磁片的檔案系統　41
1－2　磁片與主機之間的溝通：DISK I／O　47
1－3　檔案管理系統：FMS　48

1－4　DOS 摘要　53

第2章　APPLE的資料在哪裡？

2－0　資料放在哪裏：T／S表　57
2－1　檔案記要表： FDB　68
2－2　目錄表：CATALOG　71
2－3　磁片內容表：VTOC　75
2－4　檔案類型　79

第3章　I/O ERROR

3－0　修理磁片的準備工作　91
3－1　故障磁區的檢查　99
3－2　I／O ERROR的預防措施　104
3－3　DISK FULL？　109
3－4　DOS DOWN 掉怎麼辦？　112

第4章　解剖DOS

4－0　DOS 的組成　115
4－1　VTOC與目錄暫存區：哪些東西需要搬進 APPLE ？　117
4－2　檔案暫存區： BUFFER　118
4－3　第三頁向量　127
4－4　DOS 的記憶結構要點　129
4－5　FMS 的使用方法　132
4－6　RWTS的使用方法　138

4－7　RWTS的機械語言使用法　145
4－8　快速的資料搜尋法　151

第5章　DOS攔截：修改APPLE系統的方法

5－0　開機過程　163
5－1　DOS 的 I／O 攔截　165
5－2　DOS 攔截的徹底研究：KBDINCPT　169
5－3　DOS 的修改　179
5－4　消除掉的檔案如何恢復　185
5－5　修改 DOS 的命令集：自動修改程式　189
5－6　創造 DOS 的命令：TYPE 文字檔　195

第6章　RWTS徹底研究

6－0　6／2 編碼法　201
6－1　編碼程式：PRE－NIBBLE　205
6－2　RWTS的區段寫出：BLOCK WRITE　208
6－3　RWTS的區段讀入：BLOCK READ　217
6－4　RWTS主程式　222

第7章　用RAM做成的磁片——最快速的磁碟機

7－0　最快速的磁碟機：RAM－DISK　237
7－1　RAM－DISK程式說明　241
7－2　RAM－DISK的使用法　260
7－3　這樣的磁碟機是否快速？　265

附錄A　磁片記錄的基本原理

A－0　在磁片上記錄資料的方法　271
A－1　雙倍密度記錄法　275
A－2　APPLE 讀寫的硬體動作　279

附錄B　磁區讀寫程式的操作法

B－0　COPY Ⅱ PLUS 4.0　281
B－1　NIBBLE AWAY Ⅱ　284
B－2　BAG OF TRICKS 的 ZAP　287

- 本書的組合語言程式均以LISA寫成。
- 執行與磁片Ｉ／Ｏ（寫入）有關的程式時，要將寫入保護的膠紙取下。
- 爲服務讀者，本書所有程式製有磁片，請見所附郵購單說明。

本章提要

雖然學電腦從高階語言開始最簡單，但是研究電腦却要由最底層開始才能踏實，因此在第0章我們要介紹：

①INIT做些什麼？

②磁片的基本構造

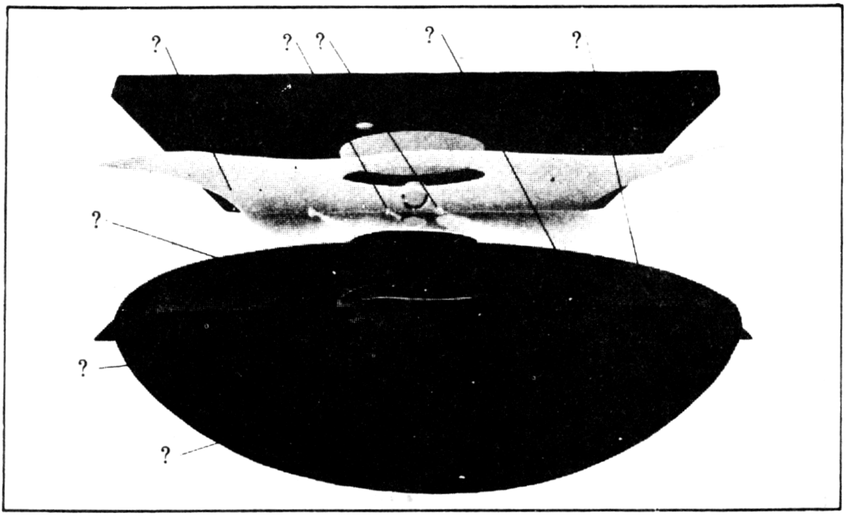
③APPLE怎樣在磁片上存放資料

④最底層的I／O：SEEKABS及READADDR程式

⑤研究磁片的基本工具程式ZAP

其中0－3至0－6節可以先略過，等到第6章再一併閱讀亦可。而如果對於磁片記錄資料的原理感到興趣，可以參照附錄A就會有滿意的答案了。

0 從頭開始：INIT



0－0 INIT？

最初，磁碟機的使用者，多半以SAVE、LOAD、RUN 以及CATALOG等指令來開始磁碟機的使用歷程，這期間，偶而也使用一些如PFS、DBASE Ⅱ、LISA等工具套裝程式。在這個層次上，磁碟機的迅速方便是勿庸置言的，但是若以提昇使用者的能力，並且對磁碟機的功能做徹底的發揮而言，則尚有一段距離。

一個磁碟作業系統（Disk Operating System 簡稱DOS）

，除了以上述的SAVE、LOAD等直接命令執行其程式儲存功能之外，更重要的是其系統執行時（Run time）的輔助儲存（Secondary Storage）能力。這種能力，使電腦的資料儲存量大大超出其主記憶體的範圍，並且資料之存取速度也令人滿意，這樣的系統，使得微電腦得以進入小型企業，並廣為使用。磁碟系統的這種優點，到底要如何來善加使用？它是否有什麼高超的潛能我們沒有發揮呢？凡事必須由根本瞭解，才能夠掌握其原則架構。所以，讓我們從INIT來踏出第一步吧！

> 請以標準DOS 磁片啓動APPLE ，NEW ⏎
> 然後放入一片全新的空白磁片，INIT HELLO ⏎
> 這張磁片，我們將作為本書的實驗說明之用。

0－1 磁片的物理結構

在進入由磁碟作業系統所管理的磁片檔案結構之前，我們有必要先瞭解磁片的物理構造，因為它是磁片最根本的形態，更進一步的系統組織全以此為基礎。

上一節我們使用的INIT指令，到底做了些什麼事情呢？原來，在一片全新的磁片上，其磁性物質是均勻分佈的，也就是說，沒有任何方向、位置標誌的，這樣的磁片我們如何能令APPLE 將資料存入磁片而又將它讀回呢？一旦資料寫入磁片後

，要在全無標誌的磁片上讀回該資料，大概就像將針丟入大海，然後又想找回來一樣的困難吧？即使找得回來，也因爲地毯式的搜尋而大大耗費了時間呢！因此，對於一張空白的磁片，我們要先將它格式化（ Formatting ），格式化的意思就是在磁片上設下位置標誌，而APPLE 只要記得存入資料的位置標誌，便能直接（不必地毯式的）到該位置讀回資料了。上一節 INIT大部份時間便是做這種格式化的工作。那麼 APPLE 的格式是怎樣的呢？

APPLE 的磁片格式（本書以DOS3.3 版本爲準），共劃分成同心圓結構的 35 個磁軌，每個磁軌又分成 16 個磁區，每個磁區又含有能存放256 個位元組的資料空間，因此共能存放 35 × 16 = 560 磁區 = 35 × 16 × 256 = 143360位元組資料，這便是一般稱呼APPLE 磁片有143 K 的原因。

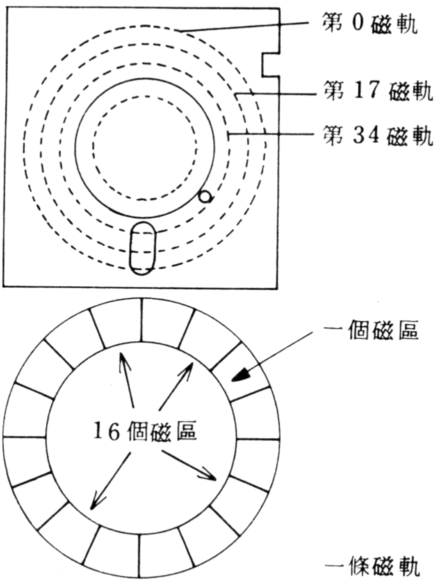


圖 0 － 0　磁片的磁軌磁區結構

上圖是 APPLE 磁片的基本構造，其基本單位是磁區，因此，往後我們稱磁片位置指的便是其磁軌磁區位置，譬如說第 3 磁軌第 5 磁區便是，並且以（$03，$05）表示之。其實磁片上還有一個第 35 號磁軌並未使用，因此眞正空間又多了 1×16×256＝4K 位元組。

磁片的 35 個磁軌及每軌的 16 個磁區分佈，可以由下圖來表示：

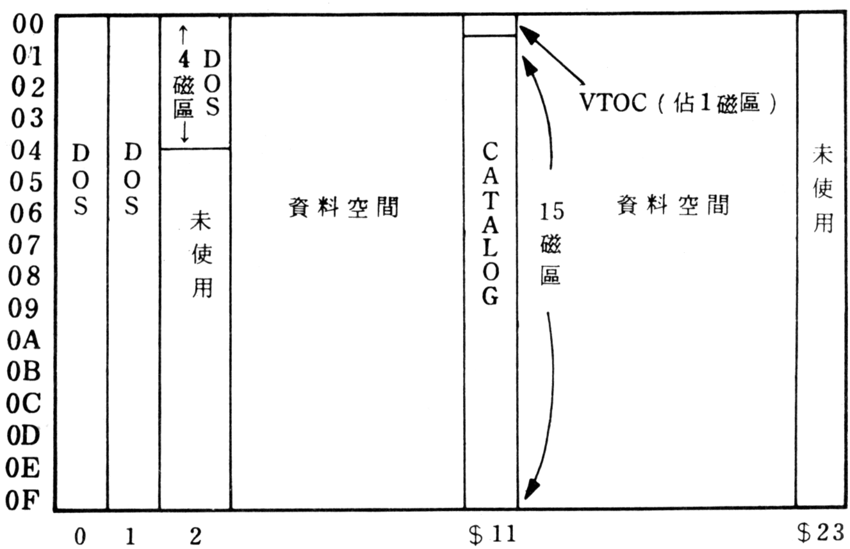
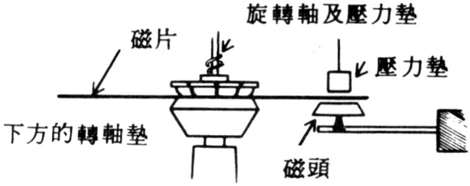


圖 0－1　磁片的磁軌磁區分佈圖（詳第1、2章）

圖中磁片最初 3 個磁軌由 DOS 使用，第 17（$11）磁軌有特殊用途（後述），第 35（$23）磁軌未使用，其它共有 30 磁軌× 16 磁區×256 位元組＝120 K 位元組可供檔案存放資料。

0－2 磁碟機的運轉：INIT的說明



（磁片在磁碟機上運轉的剖視圖）

圖0－2　APPLE的磁碟機

當我們將磁片插入磁碟機後，其側面剖視情形就如上圖所示，請注意！其讀寫頭是在磁片下方，因此資料是存放在與標籤相反的磁面上（也就是通常我們平放時與桌面接觸的那面），這樣似乎可以避免空氣中的塵埃污染，但是對於不清潔的桌面則要特別注意了

當磁碟機門關閉之後，磁片上方的轉軸壓力墊便與下方的轉軸墊一齊將磁片中央圓孔夾住，一旦轉軸轉動磁片也就跟著轉動了。磁片一轉動，磁軌上的 16 個磁區便輪翻的在磁頭上方掃過，而APPLE 便可以選擇適當的磁區來讀寫資料。除了利用磁片旋轉來選擇磁區之外，我們還要令磁頭移動來選擇磁軌，磁頭是由一個可由程式控制正反向旋轉的步級（STEPPING ）馬達來推動，步級馬達的每一相位（每旋轉一級的角度）對應於磁片上的半個軌道寬度，因此磁頭移動一個磁軌寬度，馬達要變化兩級相位。

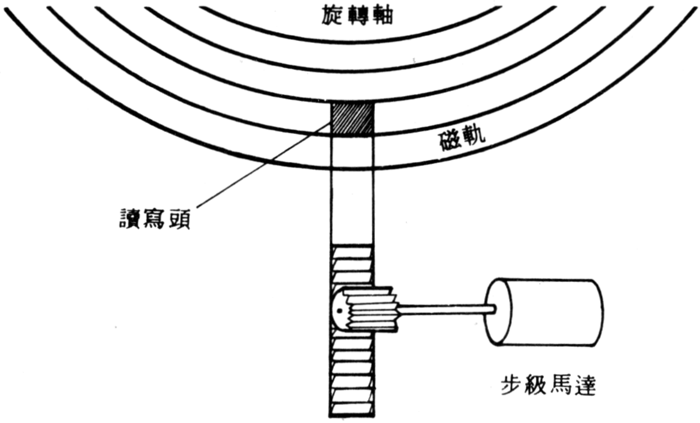
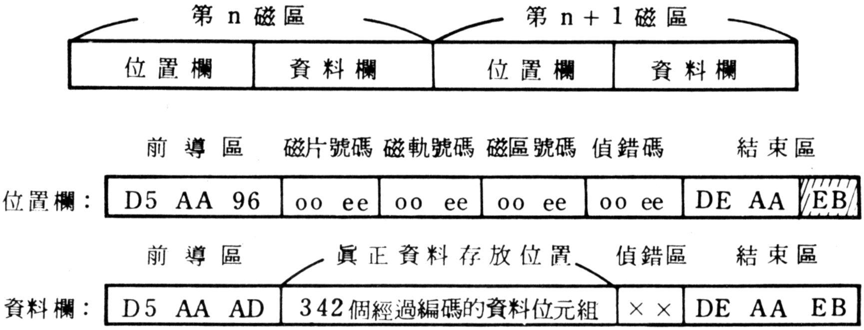


圖 0 － 3　以步級馬達控制磁頭移動的情形

知道了如何以馬達（中心旋轉馬達及磁頭步級馬達）來控制磁頭的方式之後，接著就可以來看看 INIT 的動作了。INIT 開始，首先，您會聽到一陣振動的聲音，這是磁頭歸位（第 0 號磁軌）的聲音，因爲APPLE 磁碟機是以一片塑膠片來防止磁頭跑出第 0 號磁軌以外的範圍，當磁頭到達邊界時便與膠片碰撞而造出聲音來。接著請仔細聆聽，您會發現這時磁頭正漸次的移動而造出短促的聲音來，細數這些聲音共有 34 次，正好是 35 個磁軌間的移動。在每次移動與移動間，磁頭會有短暫的停留，這些停留的作用，便是在該磁軌上造出前面我們所謂的磁片上的位置標誌，也就是將磁軌分成 16 個磁區。每個磁區上包含有位置欄與資料欄，位置欄是用來存放該磁區的位置座標值，也就是磁片、磁軌、磁區號碼等等，而資料欄則專門用以存放資料：



* 位置欄最後位元組EB在讀取時不予檢驗

圖 0 － 4　磁區位置欄與資料欄的內容

上圖中，前導區內的D5 AA 是兩個特殊位元組，專門保留來做爲每個欄的起始記號，因此 APPLE DOS 一讀到D5 AA 便知道是新的一欄的開始，至於接下來的 96 是用以指出此欄爲位置欄，而 AD 則表示是資料欄。

位置欄內的磁片、磁軌、磁區號碼均是 INIT 在格式化過程中寫入的，依此，我們才能存取資料。位置欄的偵錯區內容是將前 3 種號碼做斥或（ EOR ）運算的結果，可用爲往後檢查之用，最後的結束區是做爲同步檢查用的。至於資料欄，在前導區之後共有 342 個位元組的資料存放區間，其實這些位元組是經過編碼後的資料，眞正只有 256 個位元組而已（詳第 6 章）。

位置欄的所有資料（ V,T,S 值 ）* 均由INIT 來設定。而資料欄則由於此時磁片內容是空白，所以由 INIT 來填入 0，不

* 本書往後章節，V ＝磁片號碼，T ＝磁軌號碼，S ＝磁區號碼

過由其編碼方式；0 編碼後的值並不是 0 而是 96 。

INIT 在所有的磁軌做完上述工作後，您會聽到磁頭做大幅度移動的聲音，這是磁頭由最後磁軌（＄22）移動到第 17 磁軌（＄11）的聲音，在這個磁軌上APPLE 鍵立了磁片檔案系統的基本資料 VTOC 及目錄表（CATALOG）磁區*，請參閱圖 0 － 1 所示。接著您又會聽到磁頭再次移動到第 0 磁軌，然後將APPLE 主記憶體內的 DOS 程式寫入到由第 0 至第 2 磁軌的前 4 個磁區內（因此 INIT 時要注意主機此時的 DOS 是否完整無損，否則可能將有缺陷的 DOS 寫入新磁片而令新磁片系統產生怪異的現象）。寫完 DOS 後，磁頭又回到＄11磁軌，然後找一個磁區來開始存放 HELLO 程式檔案。這便是約 30 秒鐘的 INIT 過程，以及您常聽到的一些聲音的來源。

0－3 位元組同步法

由 INIT 造出格式化磁片後，我們便可以開始讀寫資料了，可是磁片上佈滿的是一串串的位元訊號，哪裡是一個位元組的開頭呢？在 APPLE 磁片裡，有一種專門用來做位元組同步的訊號，存放這種訊號的區域稱為同步區，同步區是位於每一位址欄、資料欄之前，用以與 APPLE 取得同步而正確的讀寫資料。

由於 APPLE 磁碟系統規定位元組的第一個（最高）位元必須是 1 ，如果一開始讀取的位元是 0 ，它會略過而等待到第一個 1 出現才讀取。利用這種特性，我們可以在一串位元組當中規則性的插入 0 ，譬如圖 0 － 5 便是在 8 個 1（FF）之後插

* VTOC 及目錄表，詳 1 、 2 章

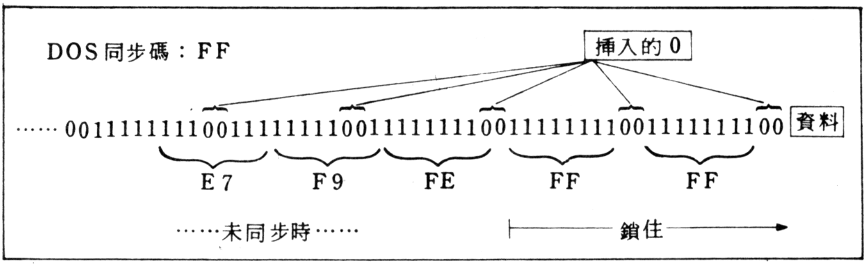


圖 0－5　位元組同步區及同步取得的情形

入兩個 0（總共是一組 10 個位元），如此一直重複。由於 10 除以 8 不能整除，而使得餘下的兩個位元便成爲下次的起始位元，重複幾次之後（最多爲 10 ÷ 2 = 5 次）一定會遇到第一位元爲那兩個額外加入的 0，這時 APPLE 便會略過而一直等待到下一個 1 的出現，而所讀到的便是 8 個 1 的 FF 並且會一直鎖住這個關係。因此眞正的資料只要安排在兩個 0 之後便可正確讀取起始位元而無錯誤，參照圖 0－5 便很容易可以瞭解了。圖 0－5 是 APPLE DOS 使用的同步位元碼，不過只要依上述原理（餘數原理），也可以造出自己的同步碼（當然最高位元要爲 1），某些磁片便是改變標準同步碼（FF）來達到防止拷貝的目的。

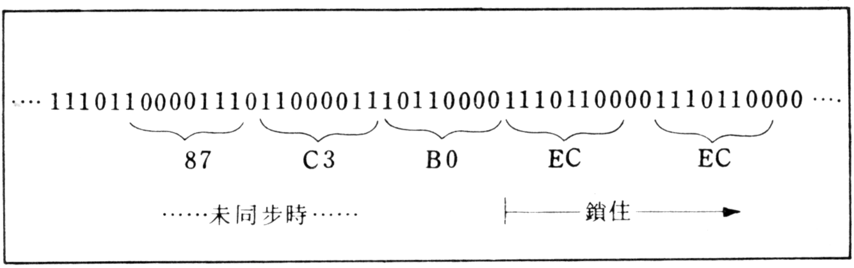


圖 0－6　以EC做同步碼

0－4 APPLE磁軌的安排

因為同步位元組的用途是要令 DOS 與磁碟機的運行取得一致，因此每次 DOS 寫下訊息之前必須先寫下同步位元組再緊接著資料，其情形就如圖 0 － 7 所示一般。磁片上最早的資料就是由 INIT 所寫下的磁片格式，磁片格式包括位置欄與資料欄，因而在二者之前都一定有由同步位元組所構成的同步區：

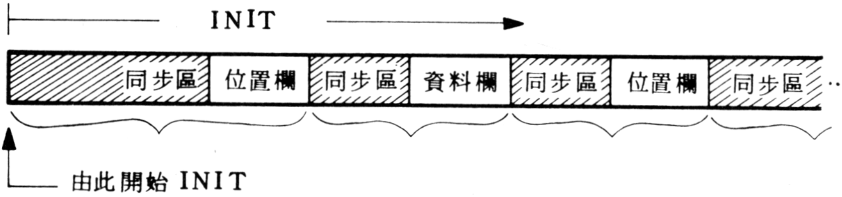


圖 0 － 7　以 INIT 造磁片的情形

所以同步區不是塡在位置欄與資料欄之間用的（ GAP ），而是做為二者的同步前導。譬如位置欄一結束之後不必是同步位元組的正確開始（第一位元），因為二者沒有關係，而雖然這時DOS讀不到正確開始，却能夠在至多 5 個位元組之內取得同步（正確的開始位置），至於其後的資料欄則一定要正確的以第一位元來接上，所以同步區是專屬於後面欄而與前面欄無關。INIT就是這樣一個同步區一個欄、一個同步區一個欄的來將一條磁軌格式化。

因為位置欄在 INIT 之後就永不更改，因此其同步區也就永不更改。但是資料欄則常要寫入新資料，所以資料欄同步區是隨

著資料更新而每次重寫的*！通常這個同步區大約由 5 ～ 10 個同步位元組組成的。至於位置欄同步區，則考慮到前面的資料欄會重寫，若磁碟機的速度不穩（變快），則可能會覆蓋到接下來的位置欄同步區，因此此區的位元組較多（14 ～ 24 個），以確保覆蓋後尚有足夠的（5 個）位元組來達到同步的目的。

當 INIT 在格式化一條磁軌時，第一次寫下的同步區要特別長（128 位元組），這是為了確保在造好 16 個磁區後，最後磁區能覆蓋此同步區而銜接上開頭的磁區，至於速度太快而覆蓋到開頭磁區本身的情形則已在設計時予以排除（那樣的磁碟機就是所謂的故障，送修可也！）。

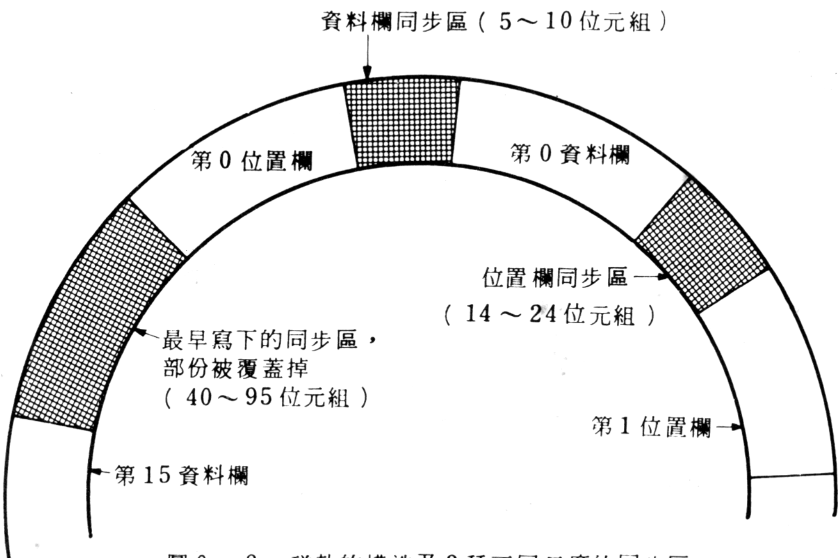


圖 0 － 8　磁軌的構造及 3 種不同長度的同步區

＊DOS的RWTS當中有一個副程式（＄B83D），每次要寫一磁區資料時，會先寫下 5 個 FF 的同步碼（中間插入 00），參考第六章。

0－5 最底層的I/O

雖然說明了INIT的動作，但是相信大家仍然對於INIT如何令磁頭移動、如何寫下資料、如何讀取訊號、如何……等感到好奇的。此處，我們就將 APPLE DOS 最深（底）一層的細節予以說明。

在APPLE 主基板上有8個SLOT，每個SLOT分配到16個記憶位址做為I／O控制用：

SLOT	解碼位址	
0	$C080 － $C08F	；共16個位址
1	$C090 － $C09F	
2	$C0A0 － $C0AF	
3	$C0B0 － $C0BF	
4	$C0C0 － $C0CF	
5	$C0D0 － $C0DF	
6	$C0E0 － $C0EF	
7	$C0F0 － $C0FF	

圖0－9 各SLOT分配的位址

APPLE DISK Ⅱ卡對於這16個位址的定義是：

記憶位址	用　　途		
$C0X0	phase 0 off	（定位）	步級馬達控制位址
$C0X1	phase 0 on	（定位）	
$C0X2	phase 1 off	（定位）	
$C0X3	phase 1 on	（定位）	
$C0X4	phase 2 off	（定位）	
$C0X5	phase 2 on	（定位）	
$C0X6	phase 3 off	（定位）	
$C0X7	phase 3 on	（定位）	
$C0X8	power down	（磁碟機 off）	
$C0X9	power up	（磁碟機 on）	
$C0XA	select 1	（選 1 號磁碟機）	
$C0XB	select 2	（選 2 號磁碟機）	
$C0XC	觸動磁頭讀寫	（I/O strobe）	
$C0XD	將資料寫入控制卡暫存器		
$C0XE	準備讀入		
$C0XF	準備寫出		

* X是磁碟控制卡的SLOT號碼加 8 。
* C0XC～C0XF用法詳第 6 章

圖 0－10　磁碟控制位址表

如果依照APPLE 的硬體解碼，及Disk slot規則，X＝6＋8＝E，所以觸動SLOT 6 上面的磁碟機的指令便是：

```
STA   $C0E9     ; 8 + 6 = E , 9 = 磁碟機ON
```

但是為了顧及控制卡可能插在不同SLOT 的情況，我們可以用X暫存器來做SLOT 索引。

```
      LDX  #$60        ; 選第 6 號 SLOT
on :  STA  $C089 , X   ; 觸動第X號 SLOT 上的磁碟機
      RTS
```

因此，使用不同的X值（ LDX #$nn ），便可以適用於不同的插座號碼。至於讀入磁片上的資料是很簡單的：

```
         LDX   #$60        ; 號6號SLOT
READIN   LDA   $C08C,X     ; 讀一個位元組
         BPL   READIN      ; 成功，則RTS
         RTS

* C08C＝讀取口位址
```

上列程式使用 BPL　READIN 是爲了保證讀取到正確的磁片位元組（跳過0位元）。由於磁片上的位元組是已經過編碼，所以讀取的值還要經過解碼後才是原來面目。

最原始的READ的程式

```
**********************************
* Routine to Read a Page of DISK *
* Data and Print Them Out ....   *
**********************************
SLOT      EPZ $0001
WAIT      EPZ $0002
CURTRK    EPZ $0004
DESTRK    EPZ $0005
DISKON    EQU $C089  )
DISKOFF   EQU $C088  ) 圖0－10的位址
DRIVEA    EQU $C08A  )
READSW    EQU $C08C  )
PRINTHEX  EQU $FDDA          ; 印出16進位數
```

```
PRINTASC EQU $FDED          ；印出ASCII字元
BUFFER   EQU $4000
*********************************
*  Set Slot Equal to Slot  6    *
*********************************
READATA  LDA #$60
         STA SLOT
*********************************
*  Turn on Drive and Wait .....  *
*********************************
         LDX SLOT
         LDA DISKON,X
         LDA DRIVEA,X
         LDA #$EF           ；等候速度穩定
         STA WAIT
         LDA #$D8
         STA WAIT+1
MWAITA   LDY #$12
MWAITB   DEY
         BNE MWAITB
         INC WAIT
         BNE MWAITA
         INC WAIT+1
         BNE MWAITA
*********************************
*  Read in Data from Disk ....   *
*********************************
         LDX SLOT
         LDY #$00
READLOOP LDA READSW,X       ；讀一位元組
         BPL READLOOP
         STA BUFFER,Y       ；暫存於BUFFER
         INY
         BNE READLOOP       ；讀256位元組
*********************************
*  Print Data on Screen ......   *
*********************************
PRLOOP   LDA BUFFER,Y       ；將讀進的資料
```

```
            JSR PRINTHEX      ;以16進位數印出來
            LDA #$A0
            JSR PRINTASC      ;印出空格(間格)
            INY
            BNE PRLOOP
*********************************
*  All done ............ Quit !  *
*********************************
ALLDONE     LDX SLOT
            LDA DISKOFF,X
            RTS
            END
```

程式0－0　讀取256位元組的程式

```
*800.843

0800- A9 60 85 01 A6 01 BD 89
0808- C0 BD 8A C0 A9 EF 85 02
0810- A9 D8 85 03 A0 12 88 D0
0818- FD E6 02 D0 F7 E6 03 D0
0820- F3 A6 01 A0 00 BD 8C C0
0828- 10 FB 99 00 40 C8 D0 F5
0830- B9 00 40 20 DA FD A9 A0
0838- 20 ED FD C8 D0 F2 A6 01
0840- BD 88 C0 60
```

程式0－0的機械碼

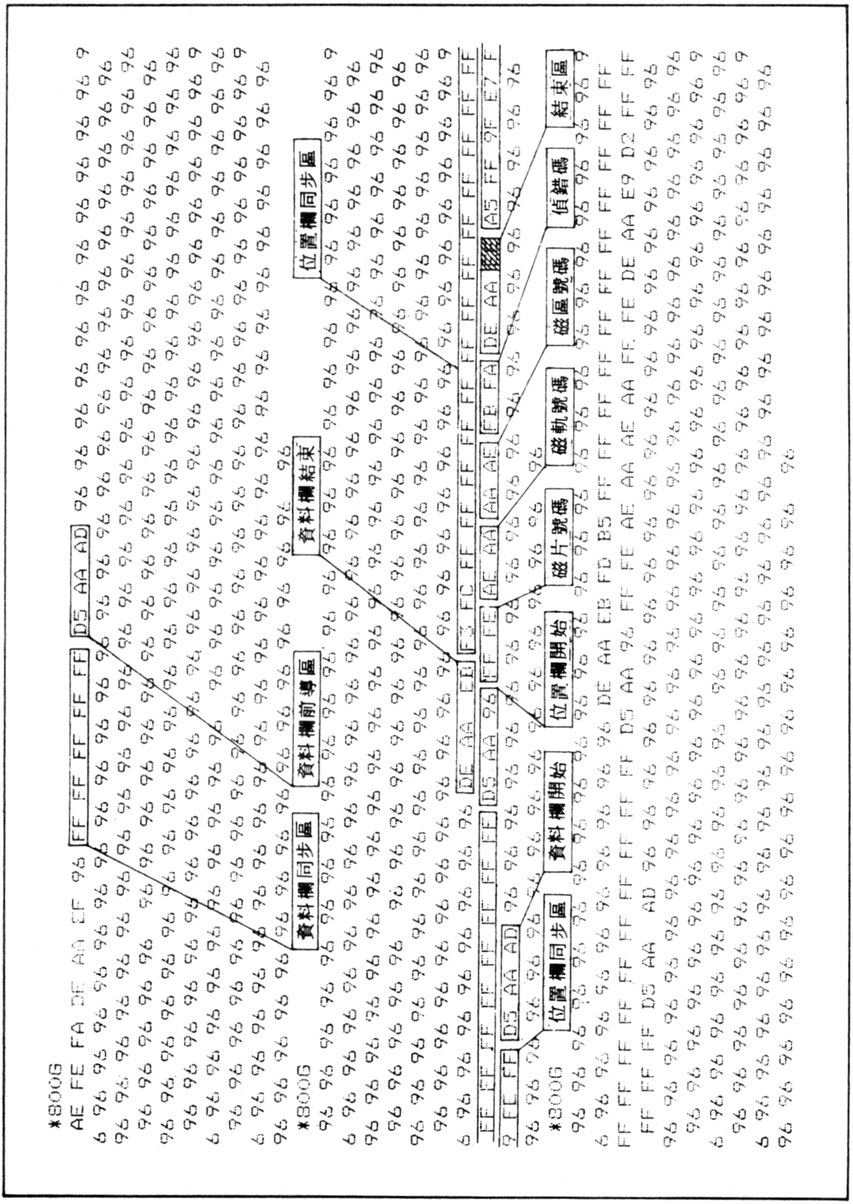


圖 0－11　以程式 0－0 讀取磁片資料的情形

上圖便是以程式 0－0 來讀取由 0－0 節所 INIT 的磁片，由於該磁片是空白內容，所以資料區多為 96（即 00 編碼後的值）。由上圖，我們可以一一來印證前幾節的說法。首先由圖 0－11 的最上面一行可以發現一串 FF 的同步訊號，然後是 D5 AA AD 的資料欄前導區，接下來的便是空白的資料區（96），由於本程式一次只讀 256 位元組，但資料欄有 342 位元組，因此再下來就看不到他種資料了。

再次 800G 來讀取資料，我們發現資料欄結束區眞的是 DE AA EB，接下來是一串同步位元組，然後便開始了下一位置欄的前導區（D5 AA 96）、磁片、磁軌、磁區號碼等等。但是，位置欄的結束區竟是 DE AA EE，最後位元組竟然不是 EB，而再次 800G 檢查其他位置欄，所有的最後位元組均似乎是不規則的亂數！為何如此呢？原來，磁碟機每次寫資料時，其最後一個位元組會受接下來的資料干擾而變化，因此要額外的再多寫一個位元組，譬如寫資料區的最後結束區時要在 DE AA EB 之後再寫入一個 FF 位元組以做為隔離（請見 RWTS 徹底研究一章），這樣 FF 雖被干擾，但 EB 却保全了。可是 INIT 寫位置欄時，其結束區並未加上 FF 位元組（DOS $BCAD～$BCC3），因此 EB 便破壞了。不過當 DOS 讀取位置欄時，却也不讀取 EB 這個位元組（詳下節），因此 EB 說是結束位元組，毋寧說是一個隔離位元組吧！

程式 0－0 是用來讀取磁片上的原始位元組，其顯示的數值是未解碼的形態。任何經過保護的磁片均能用這種方式讀取，因此是各種拷貝程式的核心。不過，單單使用這個程式，我們是無法將讀取的資料錄 'COPY' 到新磁片的正確位置上，因為，如

何判定正確的資料形態、存放位置，正是各拷貝程式的獨到之處呢！

移動磁頭的SEEKABS

或許您會發現圖0－11與你的實驗結果並不一樣，而或許您每次以800G讀出的數值均不相同吧！這是因為我們根本沒移動磁頭位置只讓磁片轉動而已，也就是說只令磁頭讀取目前所在磁軌上的資料*。但是在眞正的I／O動作時，我們是必須令磁頭依照指示來移動的，底下便是DOS 移動磁頭的程式，這個程式叫做SEEKABS（Absolutely）：

SEEKABS程式

```
*********************************
*   Track Seek Absolute Routine   *
*   Include the Timing Table...   *
*********************************
SCRATCH1  EPZ $26
SCRATCH2  EPZ $27
SCRATCH3  EPZ $2A
SCRATCH4  EPZ $2B
MONTIME   EPZ $46
*-----------------------
* DOS variables
*-----------------------
CURTRK    EQU $0478
ALTINDEX  EQU $0678
*-----------------------
* DOS entry
*-----------------------
RETURN2   EQU $B942
*-----------------------
```

```
* HARDWARE ADDRESS
*------------------------
PHASEOFF  EQU $C080          ；進入本程式時，A的值
          ORG $B9A0          ；爲目的磁軌號碼
*                            ；X的值爲插座索引
SEEKABS   STX SCRATCH4
          STA SCRATCH3
          CMP CURTRK         ；是否就是目前的磁軌？
          BEQ RETURN4
          LDA #0
          STA SCRATCH1
CHKTRACK  LDA CURTRK
          STA SCRATCH2       ；存目前磁軌號碼
          SEC
          SBC SCRATCH3       ；往前或往後？
          BEQ SEEKFIN        ；已到達，至SEEKFIN結束
          BCS STEPDOWN       ；往回走（小）
          EOR #$FF           ；往前走，變成正數
          INC CURTRK
          BCC INERTIA        ； JMP  INERTIA
STEPDOWN  ADC #$FE           ；往回走
          DEC CURTRK         ；目前位置減一
INERTIA   CMP SCRATCH1       ；由①已移動的格數
          BCC CHKLIMIT       ；   ②將移動的格數
          LDA SCRATCH1       ；   ③將移動的格數
CHKLIMIT  CMP #12            ；取最小值存入Y
          BCS DOSTEP
          TAY                ；做爲加速索引表
DOSTEP    SEC                ； phase on
          JSR STEP1          ；移動磁頭一格
          LDA WAITON,Y       ； delay
          JSR WAIT2
          LDA SCRATCH2
          CLC                ； phase off
          JSR STEP2
          LDA WAITOFF,Y      ； delay
          JSR WAIT2
          INC SCRATCH1       ；累計移動次數
          BNE CHKTRACK       ； JMP  CHKTRACK
```

查加速表

```
SEEKFIN   JSR WAIT2          ;關掉步級馬達
          CLC
STEP1     LDA CURTRK
STEP2     AND #$03           ;取低二位元變成
          ROL ;一磁軌寬      ; 0000 0XXC，C＝進位
          ORA SCRATCH4       ; 00S0 0XXC
          TAX                ;開始轉動馬達
          LDA PHASEOFF,X     ;C0q0～C0q7，q＝8＋X
          LDX SCRATCH4       ;取回 slot 號碼
RETURN4   RTS
*****************************
*      Wait subroutines     *
*****************************
WAIT1     TAX
          LDY #$A0
WAIT2     LDX #$11
WAIT3     DEX
          BNE WAIT3
          INC MONTIME
          BNE CONTINUE
          INC MONTIME+1
CONTINUE  SEC
          SBC #$01
          BNE WAIT2
          RTS
*------------------------------------------
* ACCLERATION TIME TABLE
*------------------------------------------
WAITON    HEX 013028         ;加速表
          HEX 24201E
          HEX 1D1C1C
          HEX 1C1C1C
WAITOFF   HEX 702C26
          HEX 221F1E
          HEX 1D1C1C
          HEX 1C1C1C
          END
```

請執行此程式....?

```
0300-
LDA    #$50
LDX    #$60
STA    $0478
LDA    #$00
STA    $C0E9
JSR    $B9A0
STA    $C0E8
RTS
```

程式 0－1　SEEKABS 磁頭驅動程式（機械碼見 DOS ＄B9A0～＄BA28）

這是本書第一個分析的 DOS 程式，鍵入本程式後最好馬上 SAVE 起來再 ASM ，否則可能會鍵入錯誤而令 DOS 當掉。

當然您可以用 OBJ 指令來改變目的碼存放位置，使之不直接塡入 DOS ，等到驗證無誤後，去掉 OBJ 再 ASM 一遍便可以了。若發現 DOS 可能受損，最好重新啓動，否則會有意外的損失（尤其是 INIT 、SAVE 等寫入動作）。本程式的機械碼就是DOS的$B9A0～$BA28，所以可以先用 OBJ 定在$800，ASM 後以監督程式的 800＜B9A0.BA28V 檢查是否正確，說來也是很方便的。

> LISA的使用者：在 DOS （LISA）當掉時可以用 BRK 進入監督程式，記住 $1800 位址的內含值後，再以 C600G 重新啓動LISA，最後將 $1800 的值存回，便可保住程式不被洗掉。這是因為LISA的 NEW 只不過是將程式（SOURCE）的第一位元組（ $1800 ）設成 FF ，將該值還原後程式便可取回來了。

程式0－1 基本上是很簡單的，要呼叫 SEEKABS 時必須先將磁軌號碼存入A，將插座號碼乘 16 （因硬體編碼的關係，見圖0－9）存入X，然後與目前磁軌位置（CURTRK ）比較，若不相同，則以迴圈移動步級馬達直達到目的磁軌爲止。步級馬達是以觸發圖0－10 的C0X0至C0X7 來轉動的。觸發位址若由小往大，則磁頭往內（高磁軌號碼）移，反之則往外移。

第 88 至 95 行是一組加速度延時表，由於電腦執行速度遠較馬達轉動爲快（所以才能控制），因此雖然觸動了馬達位址，馬達却需要一段時間才能轉動至指定位置。如果在馬達尚未到達指定位置，而電腦又下命令去觸動下一位址，那麼馬達便會時序混亂而無法運轉了。所以在每次觸動一個相位之後，APPLE 要等候一段時間（52、57 行），與馬達取得同步之後才可再觸發下一相位，這是步級馬達設計的要項。

APPLE 的原設計者 WOZ 考慮到雖然馬達運轉的速度很慢，但却也前後有別。因爲加速度的原因（質量慣性），剛起動時馬達速度必定較慢，可是一段時間之後速度便快了，如果APPLE能配合速度的漸快而縮短其等候時間，則磁頭移動的效率便可大爲提高了。這便是加速度延時表的來源，也是第 43 行至 48 行的目的所在，這個設計使 WOZ 又取得一項軟體專利。

使用程式 SEEKABS 時，除了設定 A 及 X 暫存器外，CURTRK 的值當然也是已知。可是打開電源啓動 DOS 時，CURTRK 的值是未知的，解決的辦法是將一個遠大於 $ 27 的值（譬如 #$ 50）存入 CURTRK，將 0 存入 A，然後呼叫 SEEKABS。依照程式 0－1 這時馬達會移動 80 個磁軌 CURTRK 才會到達 0，但是磁碟機只有 48 個磁軌範圍 *，所以磁頭早已到達邊界，這便是開機或 INIT 時磁頭振動隔板不停的發出聲音的原因了，這種做法爲的是確保磁頭一定在第 0 磁軌上，請參考程式 0－1 所附的小程式。

0－6　奇偶編碼法與READADDR

---

* 只用 35 軌

我們已經能夠移動磁頭至正確的磁軌，並且也能夠讀取資料（譬如位置欄），因此要到達任何（磁軌，磁區）位置是毫無問題了吧！可是由圖 0 － 11 位置欄的（V，T，S）值，爲什麼却看不出所以然來呢？

原來，APPLE 的磁碟系統除了不能讀取第一位元爲 0 的位元組之外，也無法讀取含有超過一對連續 0 位元的位元組。像 2A、4C、A0 這種資料是不能直接寫入磁片的，我們必須將之編碼成符合 APPLE 要求的形式來寫入磁片，而在讀取之後再予解碼回來。

奇偶編碼法

爲了避免連續的 0，最簡單的方法是將一個位元組的奇偶位元分開，各別以 1 位元間插其中，這樣就變成兩個不含有連續 0 位元的合法位元組了。使用這種編碼法，使得資料佔用空間加倍，因此較爲浪費，不過因爲方法簡單、解碼快速，所以使用在位置欄這種需要快速處理的場合是很適當的。並且由於位置欄只有十多個位元組（編碼後），因此空間浪費情形並不嚴重。

```
LDA    DATA          A←— oeoeoeoe，e＝偶 ，o＝奇
PHA                  暫存
LSR                  A＝ 0oeoeoeo
ORA    #$AA          A＝ 1o1o1o1o ，奇位元
STA    READSW，X     寫入磁片，readsw＝C0XC
延時   ………………
PLA                  A←— oeoeoeoe
ORA    #$AA          A ＝ 1e1e1e1e，偶位元
STA    READSW，X     寫入磁片
```

圖 0 － 12 奇偶編碼的方法

```
SEC                     ; C=1
LDA   odd               ; A=1o1o1o1o
ASL                     ; A=o1o1o1o1
STA   temp              ; TEMP=o1o1o1o1
LDA   even              ; A=1e1e1e1e
AND   temp              ; A=oeoeoeoe
STA   result            ; 結果=oeoeoeoe，還原
```

圖0－13　奇偶解碼的方法

```
V { FF : 1 1 1 1 1 1 1 1 ——► 1   1   1   1
  { FE : 1 1 1 1 1 1 1 0 ——►   1   1   1   0   } FE = 254
T { AE : 1 0 1 0 1 1 1 0 ——► 0   0   1   0
  { AA : 1 0 1 0 1 0 1 0 ——►   0   0   0   0   } 08 = 08
S { AA : 1 0 1 0 1 0 1 0 ——► 0   0   0   0
  { AE : 1 0 1 0 1 1 1 0 ——►   0   0   1   0   } 04 = 04

       V = 254 , T = 08 , S = 04
```

圖0－14　解讀位置欄資料的例子

DOS 的位置欄編碼是在 INIT 時由 WBYTE （$BCC4）副程式做的，而其解碼程式是在尋找磁區位置時以 READADDR 來完成的：

位置欄讀取程式：READADDR

```
!L
 ;
 *******************************
 *      Read Address Field     *
 *           (V,T,S,C)         *
 *******************************
 BUFINDEX EPZ $26
 ADRCHECK EPZ $27
 SLTINDEX EPZ $27
 SCRATCH1 EPZ $26
 CHKSUM   EPZ $2C
 SECTOR   EPZ $2D
 TRACK    EPZ $2E
 VOLUME   EPZ $2F
 BUFFPTR  EPZ $3E
 *------------------------------
 * DOS address
 *------------------------------
 RETURN2  EQU $B942
 *------------------------------
 * HARDWARE ADDRESS
 *------------------------------
 Q6L      EQU $C08C
 Q6H      EQU $C08D
 Q7L      EQU $C08E
 Q7H      EQU $C08F
 *
          ORG $B944
 *
 READADDR LDY #$FC              ; 嘗試 772 次
          STY SCRATCH1
 BYTECNT  INY
          BNE READ_D5
          INC SCRATCH1
          BEQ RETURN2           ; RETURN , C=1 ( err )
 READ_D5  LDA Q6L,X             ; 讀前導區
          BPL READ_D5
 TEST_D5  CMP #$D5
          BNE BYTECNT
```

```
                NOP
READ_AA         LDA  Q6L,X
                BPL  READ_AA
                CMP  #$AA
                BNE  TEST_D5
                LDY  #$03             ;讀3位元組(V,T,S)
READ_96         LDA  Q6L,X
                BPL  READ_96
                CMP  #$96
                BNE  TEST_D5          ;若OK,C=1
                LDA  #$00             ;測 checksum
NEXTVAL         STA  ADRCHECK
RADDR1          LDA  Q6L,X            ;讀位元組(奇位元)
                BPL  RADDR1
                ROL                   ;左移
                STA  SCRATCH1         ;暫存
RADDR2          LDA  Q6L,X            ;讀位元組(偶位元)
                BPL  RADDR2
                AND  SCRATCH1         ;位元組還原
                STA  CHKSUM,Y         ;$2F→V
                EOR  ADRCHECK         ;$2E→T
                DEY                   ;$2D→S
                BPL  NEXTVAL          ;$2C→checksum
                TAY
                BNE  RETURN2          ;若C=1(err)
TEST_DE         LDA  Q6L,X
                BPL  TEST_DE
                CMP  #$DE
                BNE  RETURN2          ;若C=1(err)
                NOP
TEST_AA         LDA  Q6L,X
                BPL  TEST_AA
                CMP  #$AA
                BNE  RETURN2          ;若C=1(err)
RETURN3         CLC                   ;C=0,正確
                RTS
                END
```

(奇偶解碼)

程式 0－2　DOS的READADDR副程式

上列程式的 51 行至 58 行便是圖 0 -- 13 的奇偶解碼法，當然其中還插入偵錯測試（CHECKSUM），35、40 及 45 行是讀取位置欄前導區位元組（D5 、AA、96 ），而 64 、69 行則用來讀取結束區的（DE AA ），這證實了前面所說的：位置欄的結束區只有兩個位元組，而 INIT 所寫下的 EB 只不過是爲了隔離而已！

0－7 下一步的工作？

到目前爲止，我們已經清楚的瞭解磁片的物理構造，並且也能夠驅動磁頭（SEEKABS）到達某特定磁軌磁區位置（由 READADDR 判斷）去讀取資料了（譬如修改程式 0 － 0 ）。綜合這些，其實您已有辦法解決絕大多數磁片保護——拷貝的問題。不過拷貝並不是我們重要的目的，我們的目的是希望能經由實務（例）與理論的途徑去瞭解微電腦作業系統的原理，而這是不太簡單的工作。在本章（及附錄A）我們學習到的是 APPLE 磁碟系統最底層的物理原理，而這只是基礎工作而已！

由下一章開始，我們要由底層的物理層次逐漸提昇到較高的邏輯層次，並且研究其間的關連。在這同時，我們也要探討本章所留下未解的一個重要問題，那就是：資料欄怎樣存放，存放的又是些什麼？其實一個磁區內絕大部份的空間是由資料欄佔用，而絕大部份使用的也是資料欄（因爲位置欄由 INIT 寫好後，就不再更改了）。往後，若不特別說明，我們一提到磁區（內容）指的便是資料欄（內容）。

資料欄內容的研究，指的並不是資料如何編碼存放的問題（這將於RWTS解說），而是當一個檔案（File）的資料存放在許多個磁區時，這些磁區之間要以何種方式來聯繫？以及檔案以何種方式來存入各個磁區？又如何取出的問題。這便是所謂的檔案結構（File structure），也就是我們接下來的主題。

基本工具

要瞭解磁碟系統，首先必須要瞭解磁片的內容。（此後，指的均為磁區資料欄的內容）但是，要怎樣來檢視磁片上的資料呢？APPLE 的磁碟作業指令雖然有讀、寫資料的功能，但我們並不知道這些資料是來自磁片上的何處，或將寫至磁片上的哪個位置！為了瞭解磁片真正的內容，我們有必要知道磁片上每一資料的詳細位置！

目前市面上有許多商品程式可以用來顯示及修改磁片上每一位元組的資料，雖然這些程式的主要功能多半是做為拷貝之用，但是其磁區修改（Sector editor）的能力，卻很適合於我們的需求。另外，一些磁片維護的程式，也一樣有這種磁區修補的功能，下列便是較常見到的程式：

-] ■ Bag of trick磁片當中的ZAP程式
- ■ COPY Ⅱ PLUS程式當中的SECTOR EDITOR
- ■ NIBBLE AWAY磁片當中的TRACK／SECTOR EDITOR
- ■ DISK PAK 6B-16的DISK ZAP
- ■ INSPECTOR
- ■ DISK FIXER

*請注意，某些稱為NIBBLE EDITOR（如LOCKSMITH、BIT

COPY及NIBBLE AWAY的TRACK/BIT EDITOR),其功能與程式0－0類似,並不是用來讀寫磁區資料的,不可混用!

本書使用的是Bag of trick的ZAP(以後簡稱ZAP),你若有任何上述的磁片均可使用,其效果對於正常的磁片結構而言是一樣的。這類程式執行時,會將你所指定的磁區內容共256位元組(已將342位元組解碼回來的數目)讀出並顯示到螢幕上來,你可以任意的改變其中的任何資料位元組,改變完畢之後並且可以寫回磁片上而達到修改磁片內容的目的。下圖便是以ZAP來讀取0－0節的實驗磁片第$11 磁軌、$0F 磁區的情形:

```
PRINT - S6,1 V$FE T$11 S$0F +$0F #0 *DU1
00 00110E000000000000000012 ............
0C 0F02C8C5CCCCCFA0A0A0A0A0 ..HELLO
18 A0A0A0A0A0A0A0A0A0A0A0A0
24 A0A0A0A0A0A0A00200000000         ....
30 000000000000000000000000 ............
3C 000000000000000000000000 ............
48 000000000000000000000000 ............
54 000000000000000000000000 ............
60 000000000000000000000000 ............
6C 000000000000000000000000 ............
78 000000000000000000000000 ............
84 000000000000000000000000 ............
90 000000000000000000000000 ............
9C 000000000000000000000000 ............
A8 000000000000000000000000 ............
B4 000000000000000000000000 ............
C0 000000000000000000000000 ............
CC 000000000000000000000000 ............
D8 000000000000000000000000 ............
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
FC 00000000                 ....
```

圖0－15 ZAP的磁區顯示幕,每行有12位元組,右邊為文字顯示區

在這個顯示幕上，橫向每列共有12 個位元組（兩個數字組成一位元組），右邊爲文字顯示區。若資料之值在 ASCII 字符範圍內，則其 ASCII 字符就顯示於此區域的對應位置上，圖中的HELLO 便是，其 ASCII 資料值是C8　C5　CC　CC　CF ，至於非 ASCII 字符的資料（控制碼等），則一律以＇·＇來表示。COPY Ⅱ PLUS 的 SECTOR　EDITOR 其顯示幕與上圖類似，不同的是每列只有 8 個位元組，當然操作手續也不相同請參考附錄B 介紹的各種程式操作法。

```
TRACK 14   SECTOR F      SLOT 6   DRIVE 1
           BUFFER 0800   DOS 16
==========================================

    0 1 2 3  4 5 6 7  8 9 A B  C D E F
   ---------------------------------------
00- 00000000 00000000 00000000 140E140D
10- 140C140B 140A1409 14081407 14061405
20- 14041403 14021401 1400030F 030E0000
30- 00000000 00000000 00000000 00000000
40- 00000000 00000000 00000000 00000000
50- 00000000 00000000 00000000 00000000
60- 00000000 00000000 00000000 00000000
70- 00000000 00000000 00000000 00000000
80- 00000000 00000000 00000000 00000000
90- 00000000 00000000 00000000 00000000
A0- 00000000 00000000 00000000 00000000
B0- 00000000 00000000 00000000 00000000
C0- 00000000 00000000 00000000 00000000
D0- 00000000 00000000 00000000 00000000
E0- 00000000 00000000 00000000 00000000
F0- 00000000 00000000 00000000 00000000
```

圖 0 － 16　INSPECTOR 的顯示幕

簡易的ZAP程式

如果您手邊沒有上述的任何工具程式，此處介紹您一個簡短而實用的BASIC ZAP程式，全部長度不過數十行，却能夠造出類如Bag of trick 或COPY Ⅱ PLUS的效果，用來配合本書實驗也是相當方便的呢！

```
0   REM ****************************
1   REM **   A Simple ZAP Program   **
2   REM ****************************
10   DIM A$(20),H$(24)
11   DATA  0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
12   FOR I = 0 TO 15: READ A$(I): NEXT I
13   DATA  1,96,1,0,6,0,32,03,0,32,0,0,1,0,0
      ,96,1,0,1,255,216,169,03,160,10,32,217
      ,3,96
14   FOR I = 0 TO 16: READ IOB: POKE 778 + I
      ,IOB: NEXT I
15   FOR I = 0 TO 3: READ DC: POKE 800 + I,D
      C: NEXT I
16   FOR I = 0 TO 7: READ INS: POKE 768 + I,
      INS: NEXT I
17   REM  DEFINE NAMES:
18 RWTS = 768:TRK = 782:SEC = 783
19   TEXT : HOME
20   VTAB 1: HTAB 1
50   INVERSE : INPUT "TRACK #:";T
60   IF (T < 0) OR (T > 34) THEN 20
70   INPUT "SECTOR #:";S
80   IF (S < 0) OR (S > 15) THEN 20
85   NORMAL
90   POKE TRK,T: POKE SEC,S
100   CALL RWTS
```

```
110  FOR I = 0 TO 21
120  FOR J = 0 TO 11
130 N =  PEEK (8192 + I * 12 + J)
140  GOSUB 500: REM  DEC -> HEX
150 H$(J) = H$
160  IF N > 127 THEN N = N - 128
170  IF N < 32 THEN H$(J + 13) = "."
180  IF N > 31 THEN H$(J + 13) =  CHR$ (N)
190  NEXT J
200 H$(12) = " "
300 N = I * 12
310  GOSUB 500: REM   DEC -> HEX
320  PRINT H$;":";
330  FOR J = 0 TO 24
340  PRINT H$(J);
350  NEXT J
370  NEXT I
380  GOTO 20
500  REM  DEC TO HEX ROUTINE
510 N2 =  INT (N / 16)
520 N1 = N - N2 * 16
530 H$ = A$(N2) + A$(N1)
540  RETURN
```

程式 0 － 3　簡易的 ZAP 程式

這個程式能夠讀取您所指定的任何磁區內容並顯示於螢幕上，請參考底下的例子：

```
TRACK #:17
SECTOR #:15
00:0011CE000000000000000012 ............
0C:0F92C8C5CCCCCFA0A0A0A0A0 ..HELLO
```

```
18:A0A0A0A0A0A0A0A0A0A0A0A0
24:A0A0A0A0A0A0A00400120B         ....
30:02DAC1D0AED3C9CDD0CCC5A0 .ZAP.SIMPLE
3C:A0A0A0A0A0A0A0A0A0A0A0A0
48:A0A0A0A0A0A00500FF0982        ....
54:C4CFD3A0C2CFD3D3A0A0A0A0 DOS BOSS
60:A0A0A0A0A0A0A0A0A0A0A0A0
6C:A0A0A0A0A0163E00FF0B04C9       .>...I
78:CED3D0C5C3D4CFD2A0B4AEB0 NSPECTOR 4.0
84:A0A0A0A0A0A0A0A0A0A0A0A0
90:A0A0A0A01A09001A0284D4D2      ......TR
9C:C1D8A0A0A0A0A0A0A0A0A0A0 AX
A8:A0A0A0A0A0A0A0A0A0A0A0A0
B4:A0A0A0A04A001F0884C9CEC9      J....INI
C0:D4A0A0A0A0A0A0A0A0A0A0A0 T
CC:A0A0A0A0A0A0A0A0A0A0A0A0
D8:A0A0A04500100384DAC1D0A0     E....ZAP
E4:A0A0A0A0A0A0A0A0A0A0A0A0
F0:A0A0A0A0A0A0A0A0A0A0A0A0
FC:A0A04700FFFF0000FFFF0000    G.....
```

*本程式的 TRACK # 及 SECTOR # 均以十進位表示

圖 0 － 17　以程式 0 － 3 讀取目錄磁的情形

由於本書往後各章所要的多爲磁片內容的觀察，因而程式 0 － 3 雖然沒有更改磁片內容的能力，說來却更實用而安全呢！並且閱讀完第 4 章後*，您便有辦法爲它加入更多（如 WRITE ）的功能了。本程式只佔 5 個磁區，載入速度很快，不久你就會發現它的好處了。

---

*程式 0 － 3 的內容，待閱讀至第 4 章即可明白

參考文獻：

Ref 1 APPLE Ⅱ DOS 使用手册

Ref 2 APPLE Ⅱ RWTS 程式

Ref 3 COPY 大全，波前，1983

本章提要

在這一章，我們要介紹電腦檔案結構的理論與方法，這些方法不只適用於APPLE Ⅱ，並且也廣泛使用於各型電腦，這是我們提昇層次的方向之一。本章的重點是：

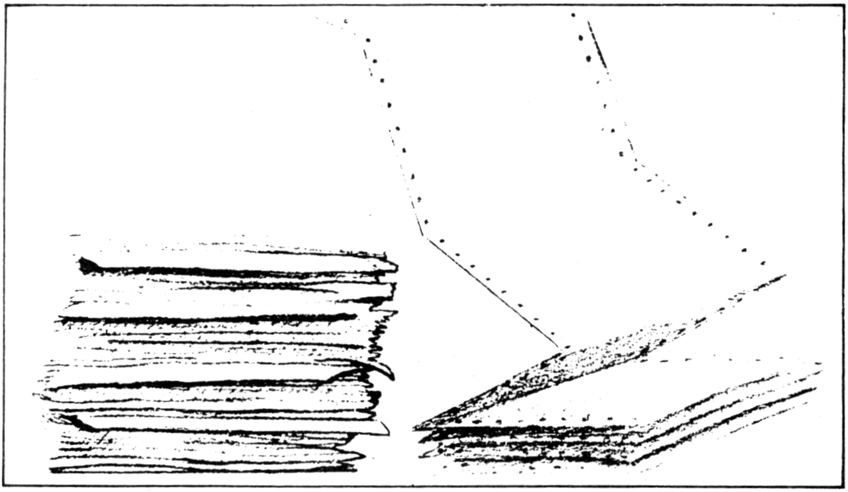
①邏輯層次的觀念

②磁片的檔案存放方式

③磁片自由空間的管理：Bit map

④DOS 的一般結構：DCI→FMS→DOIO

1 檔案系統的理論



檔案系統*是電腦研究的一門課題，深究起來也是包羅至廣且深，不過在目前單使用者（Single user）的微電腦系統內，則較為簡化而易於瞭解。因此，本章對於相關的問題將做徹底的說明，藉此機會而使我們能夠瞭解電腦檔案系統設計的方向，一方面希望由於有較廣闊的視野而使得 APPLE 系統顯得輕易而明瞭，另方面則由於基礎原理的熟習而期望對將來更新形態的微電腦系統來臨能輕易的掌握，這是我們致力的方向。

---

* 此處指的是由系統內部所管理的檔案系統，而不是應用程式設計者所建立的檔案系統，不過其技巧是類似的。

1－0 微電腦的檔案系統

雖然微電腦是一種使用者與機器之間關係極爲接近的系統，但是在操作上，如果令一般使用者直接來控制機器的每一細部工作，則必然令人因不易使用而生排斥感。因此，系統設計者便設計了一套服務程式來提高一般使用者的使用層次，也就是說，令服務程式來做各種與機器內部接觸的工作，譬如：尋找同步訊號，令磁頭讀取幾個位元組、移動多少磁軌距離等等，而使用者便昇上了高級層式，以簡單的命令就可以操作系統了。

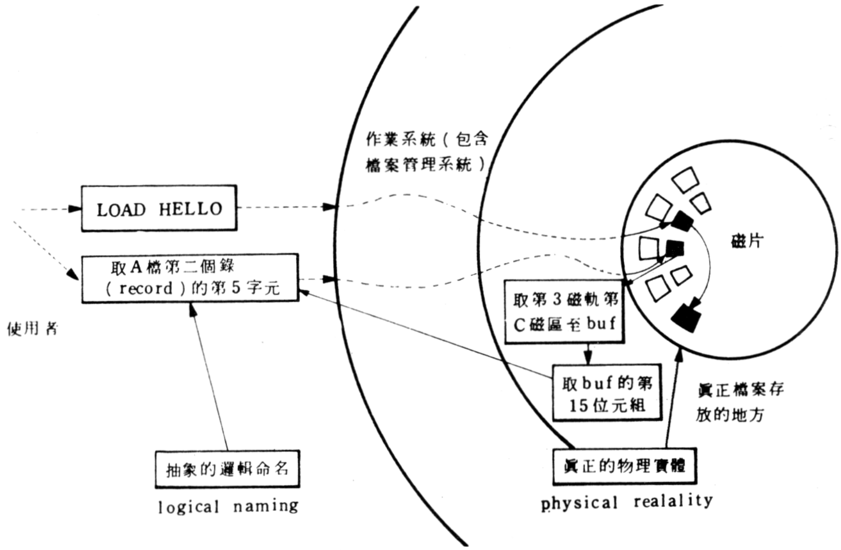


圖 1－0

邏輯層次：LOGICAL LEVEL

在電腦理論裡所謂的邏輯層次，指的是有別於物理實體的命名（ Naming ）及操作規則。譬如說：假設 HELLO 這個程式是由存在於（$05,$03）、（$05，$0E）、（$1A，$0B ）這些磁區的資料所合成的一個程式，HELLO 便是一個邏輯層次的命名，而（$05，$0E ）磁區是一個物理實體的命名*。很顯然的，對於使用者而言（指一般程式設計者），HELLO 是一個易於稱呼、記憶及使用的命名，我們只要下了：LOAD HELLO 的命令，便能夠將在（$05，$03）、（$05，$0E）、……等磁區的程式由磁片叫到主記憶體來，而不必麻煩的指出磁區的詳細號碼、位置了。這是邏輯命名的好處，至於邏輯操作呢？以 LOAD 這個命令來說，它便是一個邏輯操作的命令，其眞正的物理動作，可能是令磁碟機轉動、磁頭移位、感應訊號、將讀取結果存入主記憶體，但是使用者却只要下一個 LOAD 命令，服務程式便自動爲您做好一切事情。在 APPLE 磁碟系統使用手册上所交待的，多數便是邏輯層次的操作方法，如此使程式設計顯得容易，使用者便能夠將大部份的精神專注於他所欲解決的問題本身，而不必再費心於電腦細部動作了。

檔案管理系統

但是對於系統研究者（或是想徹底發揮電腦功能的使用者）而言，我們是有必要進一步的瞭解 LOAD 這個動作是如何進行的

* 若考慮到磁片的旋轉移位（ skewing ），則（$05，$0E）還不是眞正的物理實體。

，更希望知道 HELLO 這個程式是以怎樣的形式存於何處？

所有的這些問題是由一個介於機器實體與邏輯層次的服務程式來處理的，通常我們稱之爲作業系統（Operating System 簡稱 OS），由於 APPLE 的這個作業系統是以磁碟機運作爲主，因此稱爲磁碟作業系統（Disk Operating System，就是我們所謂的 DOS），並且因爲 APPLE 是一個較爲單純的系統，不必做複雜的多元處理（Multiprocess Management），因此其重點便集中於檔案管理工作，其實 APPLE DOS 除了指令解譯器之外，檔案相關程式竟佔了 DOS 的⅔。

回頭來看圖 1－0 其意思是：使用者面對的是高階的邏輯層次命名與指令，而這些指令交待給檔案作業系統之後，便由作業系統來執行一切實體的存取工作。通常，在邏輯層次的使用者，是不知道實體系統上的執行狀態的，而我們的研習目標，便是希望透過對於檔案作業系統的瞭解，清楚電腦內外動作的細節，進而發揮徹底的功能。

檔案作業系統的構成

檔案作業系統（往後簡稱檔案系統）的工作是將檔案資料以有效率的規則取出或存入於磁碟片上，因此：①以磁碟片而言，必須有效的控制磁片空間（free space），必須有效的存放資料（使之便於讀取、修改、增加、刪減等工作）。②以檔案管理程式與磁片之介面而言，必須建立有效率的輸出入（I／O）方法，必須使磁碟機做爲輔助記憶單元的功能得以發揮。③以使用者與檔案管理程式之介面而言：必須易於操作，使之成爲使用者高層次工作的利器，必須不因管理系統之介入而過於影響機器運

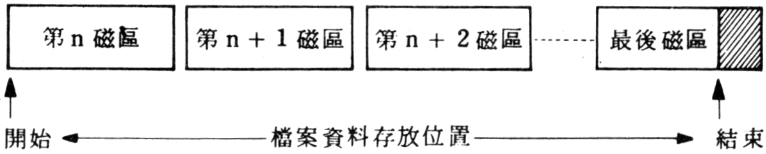
作之速度。

1－1 磁片的檔案系統

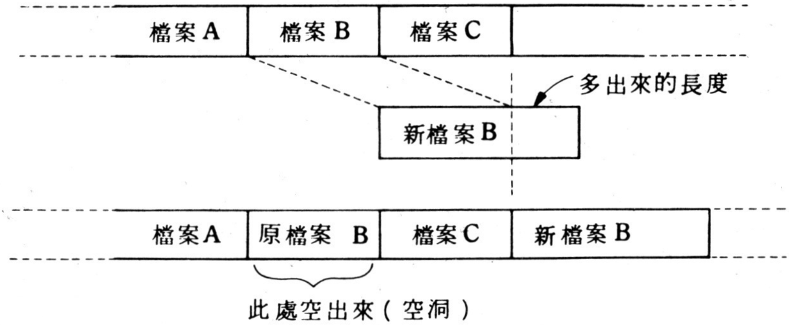
磁片存放資料的方法，稱爲磁片的檔案系統。磁片的檔案系統的研究，包含三項課目：①每個檔案本身的存放方式。②所有檔案的管理方式（一般使用CATALOG或DIRECTORY 來管理）。③磁片空間之管理，亦即記錄哪些磁區已使用、哪些磁區未使用，在較完整的系統裡甚至包含哪些磁區已損壞不可使用。

檔案存放方式

①連續存放方式（Contiguous Allocation）：

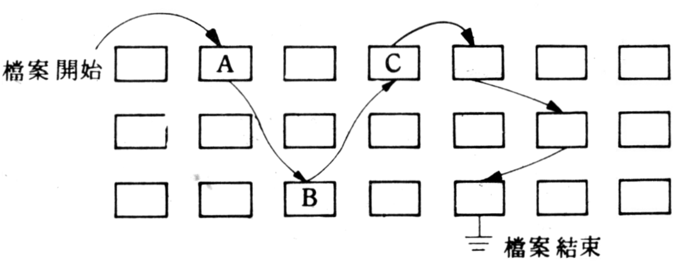


這種方式將檔案資料存放於磁片連續的磁區內，至到存完資料爲止，中間沒有跳號。其優點是讀寫速度快，因爲在讀寫過程中很少需要移動磁頭（每滿一磁軌才前進一軌），而磁頭移動是速度減慢的一大因素。這種方法的缺點是不易改變資料長度（譬如插入新資料），因爲當檔案漸多之後，每個檔案之前後均有其它檔案存在，空間已被固定，這時若要更改長度，則必須尋找一塊更大的空間，然後搬動原有資料到新位置，新資料才有可能加入。APPLE DOS 程式本身由於永不變更，所以才用這種方式存放於 0 ～ 2 號磁軌。



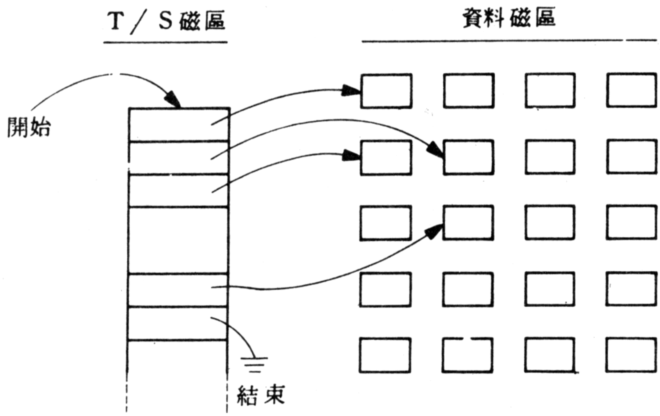
由於多次搬動，可能造出許多小空洞（因爲大空洞都可塡入程式），而使得磁片雖然還有許多空間（皆爲小空間），但却無法存入程式（若此時程式長度大於每個小空間，但小於空間總和），這種現象稱爲外在的空間散置（External Fragmentation）。這個問題可以用空間整合（Garbage collection）的方法，將小空洞合併成一整塊區間再予使用。

②連結存放方式（Linked Allocation）：



由基本資料結構我們知道使用連結（Link　）方式可以很容易的將各小空間連結起來，而成爲大空間，根本沒有外在散置的問題。連結存放法的缺點，是不適於隨機存取法，因爲譬如每次我們要讀取C磁區的資料，一定要先將磁碟機的磁頭移到A磁區，讀得其指位器的值，再移到B磁區，讀得其指位器的值，而最後才有辦法到達C磁區。因此，某些電腦系統，對於循序檔案（Sequential file），使用連結存放法，而對於隨機檔案，則使用前述的連續存放法，因爲對於連續存放的資料，只要找到檔案開頭，再空過（磁片旋轉）所要跳過的磁區（計算長度便可），便可到達目的磁區了。連結法的另一個缺點是因爲指位器分佈於各磁區，維護不易，一旦有一磁區受損，則往後各磁區便找不到而失去資料了，受損機會大的原因不只是指位器受損，也因爲磁區前後的位置欄、同步等資料受損，由於每個磁區均有這些資料（以軟式規化而言，如APPLE便是），因而損壞率增大。

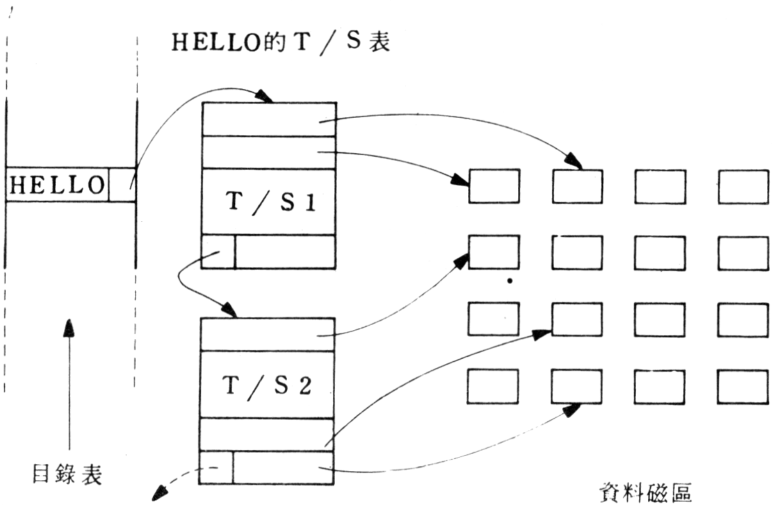
③索引存放法（Indexed Allocation）



索引存放法是爲了解決上述連結法的缺點，而將其指位器全部集中於一個磁區，這個磁區不用來存資料而只用來存指位器，指位器的內容是磁區位置，而磁區位置是以（磁軌號碼、磁區號碼）配對而成的，因此這個磁區的內容全部是磁軌、磁區（Track 、Sector ）號碼，特稱呼爲T／S磁區，以別於一般的資料磁區，一個檔案的所有T／S磁區（可能多於一個，見下一章），稱爲T／S表。採用索引存放法，克服了連結法的缺點，並且也沒有連續存放法的缺點，因爲插入資料只要在T／S表內插入一個指位器（只是位元組插入，不是磁區插入）便可以了。雖然每個檔案最後資料磁區未用完之空間可能浪費，但是這種浪費永遠小於一磁區，稱爲內在散置（Internal Fragmentation），可忽略之。APPLE 的磁碟系統使用的便是這種改良的索引存放法。

目錄表：CATALOG

由於微電腦系統多為單使用者系統，因此其目錄表很簡單，只要保存各檔案的資料便可以了，這些資料稱為檔案進入點（entry）或檔案記要（descriptor），包含檔案型式、檔案名稱、檔案長度、檔案位置等等。其中檔案位置，如果檔案資料是採用連續存放法或連結存放法，則只要指向檔案開始的磁區便可以了。如果是像 APPLE 使用的索引存放法，則就要指向T／S表的開頭磁區：



目錄表也像T／S表一樣要置於磁區內，放目錄表的磁區（可能一個以上）稱為目錄磁區。

自由空間的管理：BIT MAP

除了檔案資料的存放之外，也要注意磁片上的空間使用情形，也就是說要記錄哪個磁區已被使用，哪個磁區未使用，資料存入磁區後要將該磁區記成使用中，而檔案消除（ DELETE ）後，要將原本佔用的空間記錄成未使用，凡此種種便稱爲自由空間的管理。

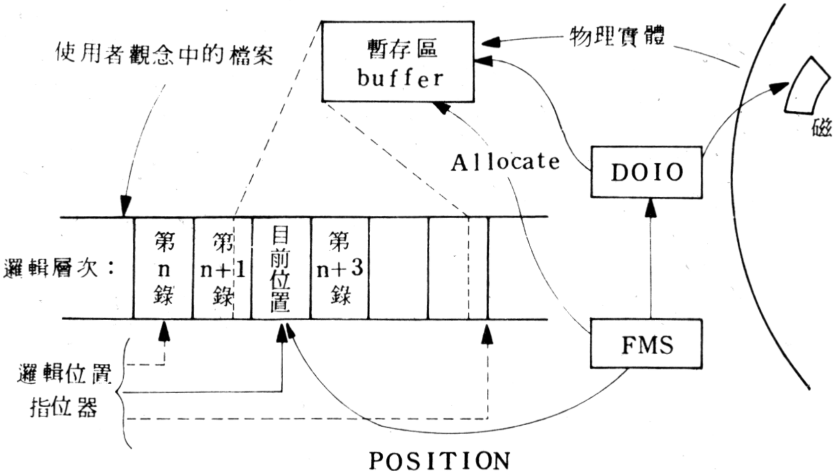
自由空間的管理方法也可以用連結或索引方法（ UNIX 系統混合二者來使用）。但是 APPLE 使用了一個較爲快速的方法，稱爲位元映射（ Bit Map ）法， 磁片上所有的磁區以一個稱爲位元陣列（向量）的表格來管理，每個磁區有一個位元與之對應，若磁區未被使用，則其對應位元爲 1 ，否則爲 0 。譬如下圖：

```
1100100000001111 ← 第 0 磁軌
1111111111111111 ← 第 1 磁軌
0000111111110101 ← 第 2 磁軌
.............................      :
.............................      :
```

位元映射表

此圖表示第 0 磁軌的第 0、1、4、C、D、E、F 磁區是未使用，第 1 磁軌全部未使用，第 2 磁軌的第 0 ～ 3 、C、E 磁區在使用中，……依此類推，APPLE 使用的是 16 磁區系統，因此用 16 個位元一組來使用，每組之間以兩個 8 位元的位元組做間格，其它系統依其設計而定。

1－2 磁片與主機之間的溝通：DISK I/O



＊由使用者觀點，邏輯位置指位器指的是檔案資料錄的一個位置，由系統（ FMS ）的觀點，位置指位器指的是 buffer 的內容。其中虛線箭頭已超出 buffer 範圍，FMS必須交換（ SWAP ）新磁區內容。

在磁片上建立了檔案系統之後，我們必須將它與邏輯層次的觀念連接起來，這種連接，包含了實際上將資料由磁片上取出而存入主機記憶體的動作（ I ／O ），以及如何將取出來的資料解釋成使用者觀念上的檔案形態等問題。在上圖的說明中，資料的存取工作是由一個叫 DOIO的程式來負責，在這個層次上，DOIO是依照 FMS 的指定來作暫存區（通常是由主記憶體特別劃分出來的）與磁片間的資料搬運工作。FMS（ File Management System，檔案管理系統）是管理檔案系統運作的程式，這個程式使得使用者能夠以邏輯層次的觀點來處理檔案，以上圖來說，當使用者讀取某一檔案的某一資料，FMS 便依磁片檔案

構造算出此資料所在的磁區，然後令DOIO將含有此資料的磁區讀到暫存區（Buffer）來，並且透過GET、INPUT或PRINT指令來將資料與使用者程式溝通，如果下一個讀寫動作的對像，仍然是暫存區的資料，則直接到暫存區讀寫便可以，如果是超過暫存區所含資料的範圍（圖中虛線箭頭的邏輯位置指位器所指的便是），則FMS便令DOIO將目前在BUFFER內的資料寫回磁片上對應的磁區，然後將新資料所在的磁區搬入BUFFER內，這時使用者（的程式）便可以讀寫這個新資料了。

由以上的說明，我們知道實際上只有FMS、DOIO、磁片、BUFFER等存在，圖中邏輯層次的檔案其實並不存在的，檔案實體只有現存於BUFFER的部份是在主機內，其它部份還在磁片上呢！通常DOIO所做的是以磁區爲單位的I／O，稱爲Block I／O，在APPLE內DOIO是由一個叫RWTS的程式來處理的，一次共搬動256個位元組（一個磁區），在Unix系統DOIO是一個中斷服務程式，每次搬動512個位元組（一個磁區），而CP／M系統的DOIO則是由BIOS部份來達成的（APPLE CP／M的BIOS包含了部份RWTS）。與Block I／O不同的I／O叫做Character I／O，每次只有一個字元（Character）輸出入，譬如APPLE的鍵盤輸入及MODEM均是。

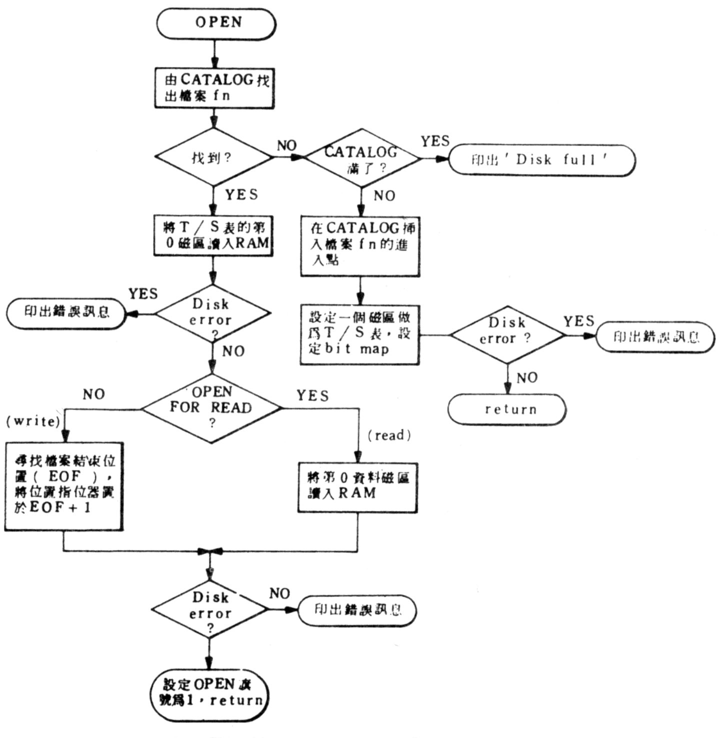
1－3 檔案管理系統：FMS

FMS的職責已由前節略述，它的工作除了調度DOIO來執

行低階的資料搬運工作之外，還要令使用者能在較高階層操作檔案。使用者如何操作檔案呢？常見的是CREATE、OPEN、READ、WRITE、CLOSE 等，請見下列各例＊：

OPEN

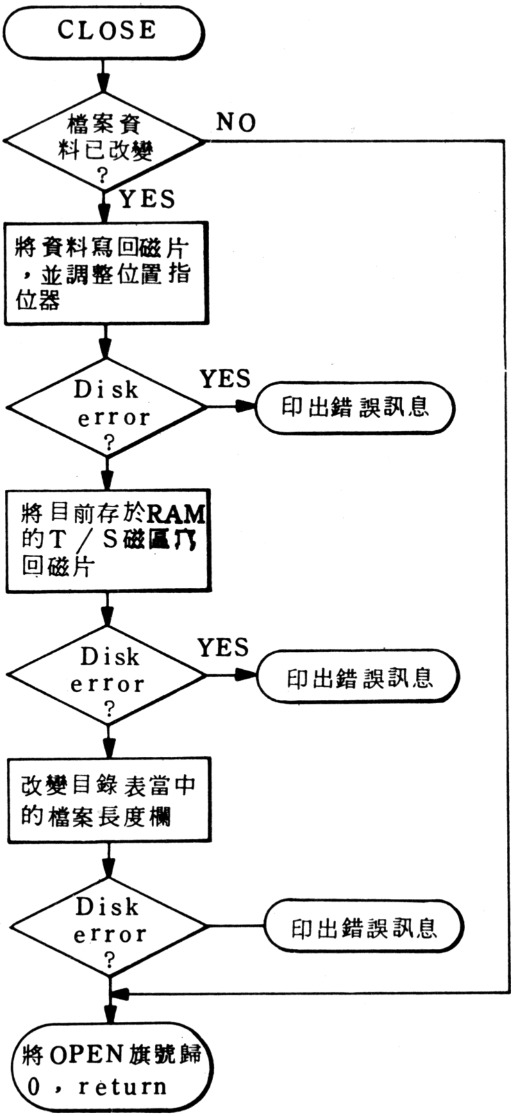
有些系統將 CREATE 與OPEN合併在一起（如APPLE DOS），有此則分開（如CP／M 的OPEN System call），以使用的經驗來說並無多大差異：



OPEN的流程（包含CREATE）

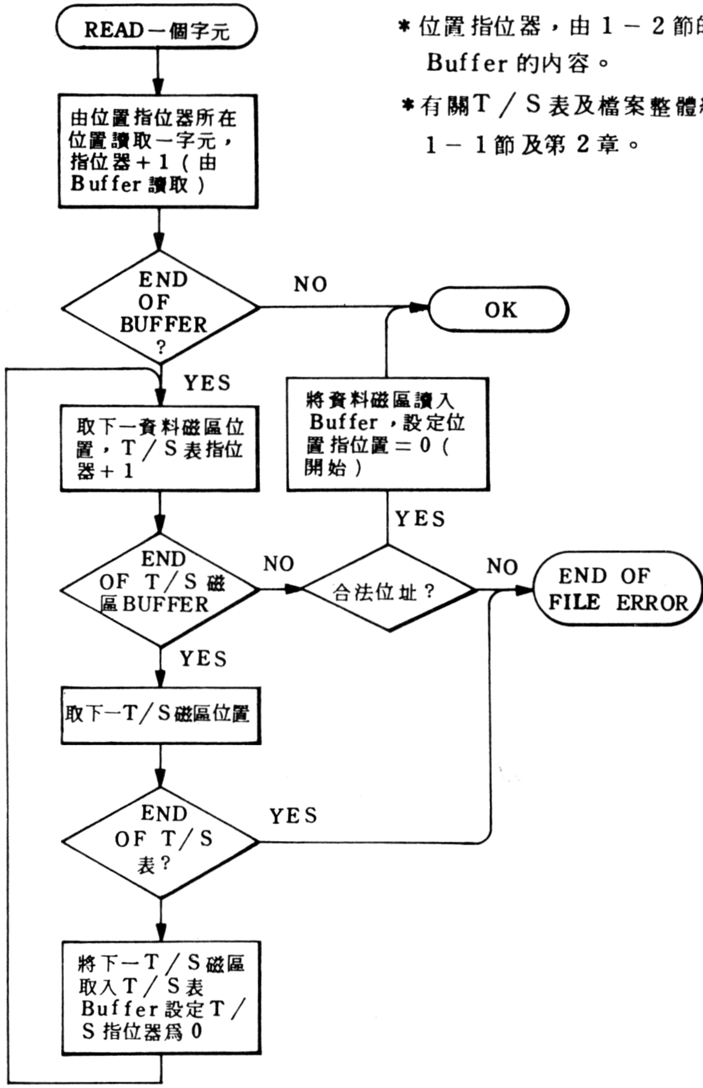
＊此處只爲舉例，細節則隨電腦機種差異而有所不同。

CLOSE



CLOSE的流程

READ

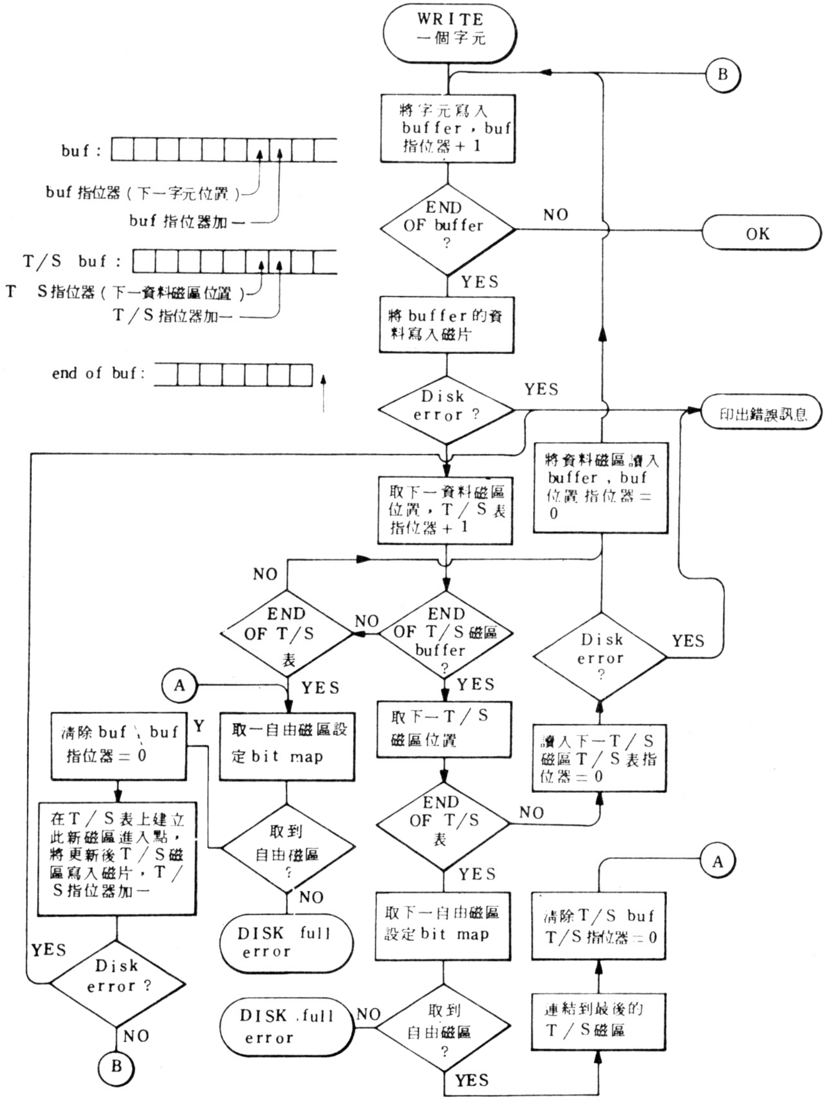


＊位置指位器，由1－2節的圖示是指向Buffer的內容。

＊有關T／S表及檔案整體結構請參考1－1節及第2章。

READ的流程，由使用者的觀點READ是以一個字元爲單位。

WRITE



WRITE 流程

1-4 DOS摘要

一個以磁碟爲主的（Disk based）系統，主要的功能是在於檔案的操作，其靜態部份爲磁片上的檔案存放規則，而動態的部份則在於檔案的搬動（DOIO）以及邏輯與實體層次的轉換（FMS）。有的系統（APPLE）再將用來解譯OPEN、WRITE、……等高階指令的解譯器由FMS中分離出來，稱爲命令解譯器（DOS Command Interpreter 簡稱DCI）。其各部功能依層次高低列如下表，本書往後各章即依此方向來研究APPLE的DOS結構。

磁片：

VTOC	Bit Map 等整體資料
目 錄 表	所有檔案的記要（進入點）列表
T／S 表	檔案資料磁區的指位器列表
資料磁區	存放檔案資料值

DOS：

層　次	動　作
使用者程式	OPEN、READ、……
DCI	呼叫FMS
FMS	讀寫buffer內容，若更新或超出buffer內容，則呼叫DOIO
DOIO	交換或重疊buffer與磁片上資料

參考文獻：

Ref 1 R. Fritzson, Data on Disk: Implementing File Systems, Microcomputing, Jan '81, P111

Ref 2 Shaw, The Logical Design of Operating System, Prentice-Hall, 1974

Ref 3 Peterson & Silberschatz, Operating System Concepts, Addison-Wesley Co., 1983

Ref. 4 Ritchie & Thompson, The Unix Time-Sharing System, Bell system Technical Jounal, Vol. 57, No. 6, P. 1905

Ref 5 Greg Tibbetts, Softcard symposium, Softalk, August／83, P227

本章提要

在一個磁碟系統當中，檔案系統（ file system ）是佔了相當重要的地位，尤其是 APPLE 這種單使用者（ single user ）系統，由於其構造簡單（只有命令解譯器），沒有複雜的多重處理（ multiprocessing ）及分時工作（ timing sharing ）等核心（ kernel ）部份，因而使得檔案系統成爲整個磁碟系統的重心角色，其檔案處理程式足足佔了整個系統程式約 2 ／ 3 的比例。

因此由檔案系統來瞭解 APPLE的磁碟作業系統，是最省時省力並且可以清楚掌握重點的方法，這是我們研究 APPLE DOS 的最佳進入點。進入本章，您會熟悉：

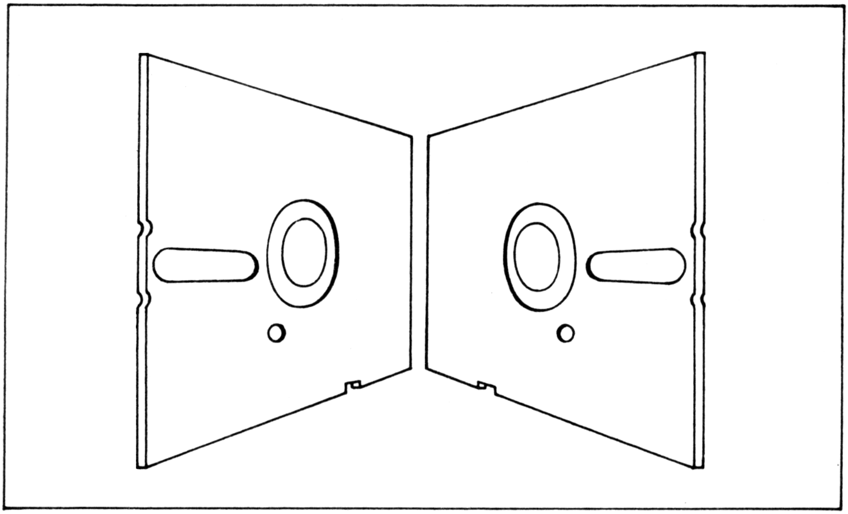
①T ／ S 表的構造。

②目錄表與 FDB 的安排。

③VTOC 與位元映射表（ Bit map ）。

④各種檔案在磁片上存放的型式。

2 APPLE的資料在哪裏？



2－0 資料放在哪裏：T/S表

由第 0 章我們瞭解了以 INIT 所造出來的磁片格式，而由第 1 章知道了檔案設計的法則，現在讓我們來看看 APPLE 的磁片檔案是怎樣的一種結構：

磁軌＿磁區表(T/S LIST)

資料，依照 0 － 1 節的說明，當然是存放到磁區裏頭了，由於磁區是檔案系統的最小讀寫單元，因此一個檔案是包含了一個

或多個磁區，這種存放資料的磁區叫做資料磁區（Data Sector）。到底我們所寫入的資料是存放到哪一個磁區了呢？APPLE 要由哪個磁區來讀（找）回資料呢？由於 APPLE 使用的是索引存放法（見第一章），因此要由T／S表來指出一個檔案其資料磁區的位置（第幾磁軌，第幾磁區）：

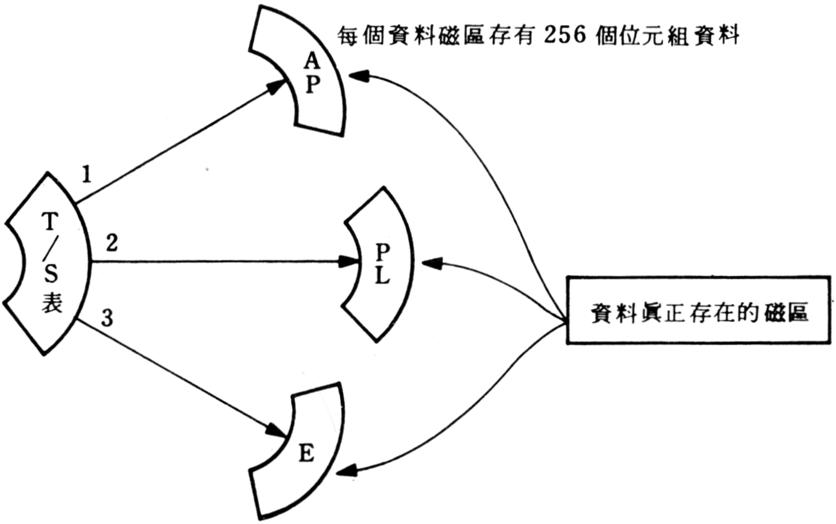


圖 2－0　T／S 表依資料順序來指出資料磁區位置的情形

上圖中，APPLE 的T／S表的資料磁區指位器是按檔案資料的先後次序來排列的，因此，要讀取檔案的第一個資料位元組，必須到圖中T／S表第 1 號指位器所指的資料磁區的第一位元組位置去讀取。程式讀取檔案時，是依T／S表的次序來讀取每個磁區的。

請以第 0 章所介紹的工具程式 ZAP （或 COPY Ⅱ PLUS、

INSPECTOR 、……等均可）來讀取我們在0－0節所INIT的實驗磁片，BRUN ZAP之後，放入實驗磁片，鍵入 R12，0F（COPY Ⅱ PLUS的使用者，請進入SECTOR EDITOR之後，鍵入R及磁軌、磁區號碼（12,0F）即可），螢幕上每會出現如下的情形：

```
00 000000000000000000000000  ............
0C 120E00000000000000000000  ............
18 000000000000000000000000  ............
24 000000000000000000000000  ............
30 000000000000000000000000  ............
3C 000000000000000000000000  ............
48 000000000000000000000000  ............
54 000000000000000000000000  ............
60 000000000000000000000000  ............
6C 000000000000000000000000  ............
78 000000000000000000000000  ............
84 000000000000000000000000  ............
90 000000000000000000000000  ............
9C 000000000000000000000000  ............
A8 000000000000000000000000  ............
B4 000000000000000000000000  ............
C0 000000000000000000000000  ............
CC 000000000000000000000000  ............
D8 000000000000000000000000  ............
E4 000000000000000000000000  ............
F0 000000000000000000000000  ............
FC 00000000                  ....
```

＊隨著本書繼續進行，由於多次檔案存取（譬如程式錯誤，修改後再存入磁片等等），可能你的結果會與書本的磁區資料內容有所不同，但並不影響基本原理的解說。

這便是一個T／S表，表上大部份的值爲0，只有第（$0C,$0）兩個位元組的值爲（$12,$0E），這便是T／S表的一對指

位器，指向第 12 磁軌的第 E 個磁區，意思是說檔案的資料是存於第 12 磁軌第 E 磁區上面，那麼到底是哪一個檔案呢？將實驗磁片 CATALOG 之後，發現只有 HELLO 一個檔案，因此（$12,$0E）磁區存的當然是 HELLO 的內容了。可是實驗磁片是 NEW 以後再 INIT 的，因此 HELLO 的內容應該是空的才對，使用 ZAP 來讀取（$12,$0E）磁區，結果如下：

```
00 030000000000000000000000  ............
0C 000000000000000000000000  ............
18 000000000000000000000000  ............
24 000000000000000000000000  ............
30 000000000000000000000000  ............
3C 000000000000000000000000  ............
48 000000000000000000000000  ............
54 000000000000000000000000  ............
60 000000000000000000000000  ............
6C 000000000000000000000000  ............
78 000000000000000000000000  ............
84 000000000000000000000000  ............
90 000000000000000000000000  ............
9C 000000000000000000000000  ............
A8 000000000000000000000000  ............
B4 000000000000000000000000  ............
C0 000000000000000000000000  ............
CC 000000000000000000000000  ............
D8 000000000000000000000000  ............
E4 000000000000000000000000  ............
F0 000000000000000000000000  ............
FC 00000000                  ....
```

*空白的BASIC程式長度爲3： 00 00 00

圖 2－1 HELLO程式的空白內容

結果竟然不是空白，爲何？（後述，請一定要依 0－0 節的方法 INIT一張磁片，否則會有以前消去（DELETE）的檔案內容殘

餘著，實驗結果會有所不同），現在將實驗磁片修改成：

```
NEW
10  REM  THIS IS A TEST
20  PRINT "ABC"
SAVE HELLO
```

現在，HELLO的內容應該不再是空白了，再以 ZAP （或 COPY Ⅱ PLUS ）讀一遍（ $12, $0E ）磁區

```
00 230016080A00B22054484953  #.....2 THIS
0C 204953204120544553540021   IS A TEST.!
18 081400BA2241424322000000  ...:"ABC"...
24 004700000000000000000000  .G..........
30 000000000000000000000000  ............
3C 000000000000000000000000  ............
48 000000000000000000000000  ............
54 000000000000000000000000  ............
60 000000000000000000000000  ............
6C 000000000000000000000000  ............
78 000000000000000000000000  ............
```

圖2－2　HELLO程式的內容

上圖便是 HELLO 程式存在磁片的內容，你可以放入實驗磁片重新開機，然後鍵入：

```
]CALL-151

*800.84F

0800- 00 16 08 0A 00 B2 20 54
0808- 48 49 53 20 49 53 20 41
0810- 20 54 45 53 54 00 21 08
0818- 14 00 BA 22 41 42 43 22
0820- 00 00 00 00 BD 4D 08 85
0828- 3D CE FF 08 AD FE 08 85
0830- 27 CE FE 08 A6 2B 6C 3E
0838- 00 EE FE 08 EE FE 08 20
0840- 89 FE 20 93 FE 20 2F FB
0848- A6 2B 6C FD 08 00 0D 0B
```

圖 2－3　HELLO程式在RAM裡頭的內容

比較圖 2 － 2 與圖 2 － 3，我們發現在主機記憶體內由 $801 至 $823 的 HELLO 程式全部內容，已被搬至磁片（ $12,$0E ）磁區的第 $02～$24 位元組內，以圖 2 － 2 來說明，其中第 2 、 3 位元組的（ $16,$08 ）指的是下一行程式的起始位址， $0A 是行號 10 的意思， $B2 是指令 REM 的代碼， $20 是空格，接下來是 THIS IS A TEST 的 ASCII 值，接下來的 00 表示行號間格，（ $21,$08 ）指的又是下一行號起始位置， $14 代表的是行號 20 ， $BA 是 PRINT 的指令代碼， $22 是雙引號〃， 41,42,43 是 ABC 的 16 進位 ASCII 碼，程式以連續 3 組 00 爲結束（詳見第一册）。圖 2 － 3 $824 以後的雜亂內容是 DOS 最初讀進 RAM 而未被移位（ relocate) 的碼，此處不予理會。至於圖 2 － 2 的第 $00 位元組的值（ $23 ）是什麼意思呢？原來圖 2 － 2 的第 0 、 1 兩個位

元組存的是程式的長度（低位元組在前高位元組在後），本程式由$2～$24共長 35（$23）位元組，因此長度爲 $0023，存入第0、1兩位元組內便成爲（$23,00）。這便是圖2－1的 HELLO 不是空白檔案的原因了，因爲有3個0位元組做爲結束，檔案長度必須記成 03，因而佔用了該磁區。

將上面的實驗做成結論是：每個檔案有一個T／S表，T／S表上以指位器（第一位元組爲磁軌號碼，第二位元組爲磁區號碼）來指出資料眞正存放的磁區位置，資料眞正存放的磁區叫做資料磁區，T／S表所在的磁區叫做T／S磁區。如果一個資料磁區（256位元組）不足以裝滿所有檔案資料，則T／S表便增加一組指位器來，以指向另一新的資料磁區以繼續存放資料。因此T／S表上面的指位器是跟著資料磁區而順次伸延著：

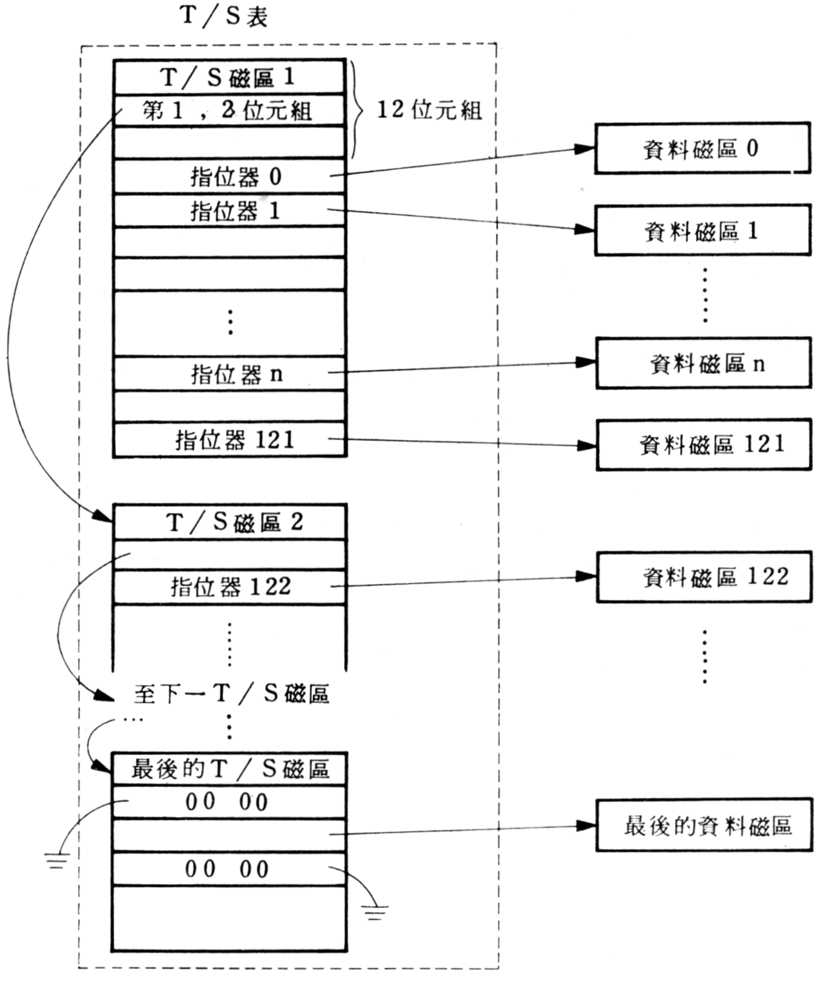


圖2－4　T／S磁區與資料磁區的構造

由於 APPLE T／S 磁區的指位器是由第 0 C位元組開始的，每組指位器佔 2 位元組，因此一個T／S磁區共有（256 －12）／

2 = 122個指位器可以指向122個資料磁區，如果檔案長度太長122個資料磁區容納不下，則便再需要有另一個T／S磁區來存放指位器以指到更多的資料磁區。

由於需要額外的T／S磁區，因此，也要有另外的指位器來指向（稱做T／S磁區指位器，不同於資料磁區指位器），每個T／S磁區的（$01,$02）位元組便是下一T／S磁區的指位器，$01 的值是磁軌號碼，$02 的值是磁區號碼。另外，在T／S磁區的（$05,$06）兩個位址上存的是該T／S磁區第一號指位器所指的資料磁區在整個檔案內的磁區號次，譬如圖2－4的T／S磁區2所指的第一個資料磁區號碼是122，因此T／S磁區2的（$05,$06）位元組的值便是（122=$7A,$00）。T／S磁區除了（$01,$02）及（$05,$06）之外在位址$C之前的位元組均不使用。有關T／S表的安排請詳圖2－4。為了證明我們的說法，現在請放入實驗磁片，重新開機，然後：

```
 NEW
 DELETE HELLO
]BSAVE HELLO,A$800,L$7FFF

]CATALOG

 B 131 HELLO
```

HELLO 這個程式已經變成佔有131個磁區的B型檔了。使用ZAP 程式讀取（$13,$0F）磁區，這是T／S表的開始磁區（為何現在是（$13,$0F）而不是以前的（$12,$0F）？）

```
00 001B0600000000000000000000  ............
0C 130E130D130C130B130A1309  ............
18 130813071306130513041303  ............
24 1302130113001 40F140E150F  ............
30 150E150D150C150B150A1509  ............
3C 150815071506150515041503  ............
48 150215011500160F160E160D  ............
54 160C160B160A160916081607  ............
60 160616051604160316021601  ............
6C 1600170F170E170D170C170B  ............
78 170A17091708170717061705  ............
84 170417031702170117001 80F  ............
90 180E180D180C180B180A1809  ............
9C 180818071806180518041803  ............
A8 180218011800190F190E190D  ............
B4 190C190B190A190919081907  ............
C0 190619051904190319021901  ............
CC 19001A0F1A0E1A0D1A0C1A0B  ............
D8 1A0A1A091A081A071A061A05  ............
E4 1A041A031A021A011A001B0F  ............
F0 1B0E1B0D1B0C1B0B1B0A1B09  ............
FC 1B081B07                  ....
```

圖2－5　長檔案的第一T／S磁區（$13，$0F）

由圖2－5可以看出其T／S磁區指位器（第1，2位元組）的值為（$1B,$06），表示本T／S表的下一個T／S磁區是在第 $1B 磁軌的第 $06 磁區，而第5、6位元組的值為0表示（$0C,$0D）指位器所指的磁區（第 $13 磁軌的第 $0E 磁區）為 HELLO 這個檔案的第0號磁區（最開頭的磁區），HELLO 所有的資料便是存在於圖2－5由 $0C ～FF 所指的122 個磁區內。

但是 HELLO 程式共有 131 個磁區（由 CATALOG:B 131 HELLO 可以看出），其它的 131 － 122 ＝ 9 個磁區是存於何處

呢？這就要查T／S表的下一個磁區了（即$1B,$06）。使用 ZAP 程式來讀取（$1B,$06）其結果如下：

```
00 0000000000007A0000000000000  .....Z......
0C 1B051B041B031B021B011B00     ............
18 1C0F00000000000000000000     ............
24 000000000000000000000000     ............
30 000000000000000000000000     ............
3C 000000000000000000000000     ............
48 000000000000000000000000     ............
54 000000000000000000000000     ............
60 000000000000000000000000     ............
```

圖 2－6　HELLO檔案的下一個T／S磁區

在這個磁區內，我們發現，其第 1 、 2 兩個位元組的值為 0 ，表示T／S表到此結束，沒有下一個 T／S磁區。而第 5 、 6 位元組的值為（$7A,$00），正如前面所說的，表示本T／S磁區所指的資料磁區是由 HELLO 檔案的第 7A = 122 號（由 0 算起）磁區開始，然後由 $0C 至 $18 共有 7 組指位器，共指向 7 個資料磁區！可是，剛才我們不是說還有 9 個磁區嗎？另外兩個哪裏去了呢？原來，另外兩個是T／S表本身佔用的磁區，也就是說，以 CATALOG 所列出來的目錄，其檔案名稱之前的數字，是代表該檔案所佔用的磁區數，包含了T／S表本身，而不只是資料磁區而已。這便是為什麼我們第一次 INIT HELLO 時，雖然其內容為 0 ，但卻佔用了兩個磁區，原來一個是T／S表磁區，而另外一個是資料磁區（該例中，只存長度而已）。

T/S磁區使用摘要

位元組	用途
00 – 00	未使用
01 – 02	下一T/S磁區指位器
03 – 04	未使用
05 – 06	第一指位器所指資料磁區的號次
07 – 0B	未使用
0C – 0D	第0資料磁區指位器
0E – 0F	第1資料磁區指位器
⋮	⋮
FE – FF	第121資料磁區指位器

2－1 檔案記要表：FDB

但是，你會問：APPLE 怎麼知道T/S表是在（$12,$0F）或（$13,$0F）呢？我們可以由T/S表的指位器去找出資料磁區的位置然後讀取資料，但是T/S表又在哪裏呢？由第一章我們知道，T/S表的位置可由目錄表找到。

基本上，磁片上的檔案是以一個叫做檔案記要（File Descriptor Block簡稱 FDB）的表格來控制的，這個表格上便記載著該檔案的T/S表位置。每個檔案記要包含2位元組的T/S表指位器、1位元組的類型與上鎖旗號、30位元組存放檔案名稱、2位元組存放檔案的磁區個數（即磁區長度），共2＋1＋30＋2＝35位元組。請詳下圖說明：

T／S表指位器	類型與上鎖旗號	檔案名稱（若未用完則填入空白）	長度
2位元組	1位元組	30位元組	2位元組

至於檔案記要表的內容所代表的意義，現在解釋如下：

第一欄：T／S表指位器，指到T／S表所在的位置（磁軌、磁區號碼）。

第二欄：上鎖旗號及檔案類別欄，由於 APPLE 的檔案分成很多類型，因此以此欄來加以區分，列表如下。如果檔案上了鎖（ LOCK ），則本欄的最高位元為 1 （ ≥ 80 ），否則為 0 。

上　鎖	未上鎖	類　　　　別
80	00	文字檔
81	01	整數 BASIC 程式檔
82	02	APPLESOFT 程式檔（ A檔 ）
84	04	二進位檔（ B檔 ）
88	08	S檔
90	10	R檔（ TOOL KIT ）
A0	20	新A型檔
C0	40	新B型檔（ LISA的L檔 ）

由於最初的 APPLE 只有T，I，A，B四種類型的檔案，但是一個位元組共有 256 種表示法，除了最高位元組做為上鎖旗號之外，尚餘 7 個位元組共有 128 種檔型，即使每一位元（ bit ）代

表一種檔案類型的話（這種方法較易於程式編寫，速度也快），則也可有 7 種不同的檔型，再加上 7 個位元全為 0 的情形共 8 種，這便是上表的由來。表中 S，R，新 A 型，新 B 型，在APPLE 原來的檔案系統內並未定義，但是在 APPLE 後來的 DOS TOOL KIT 的磁片中，已建立了 R 型（ 10 ）的檔案，而後來的LISA 組譯程式系統又建立了 L 型（ 40 ）的檔案，而某些特殊系統也各自有它們自行定義的檔案。

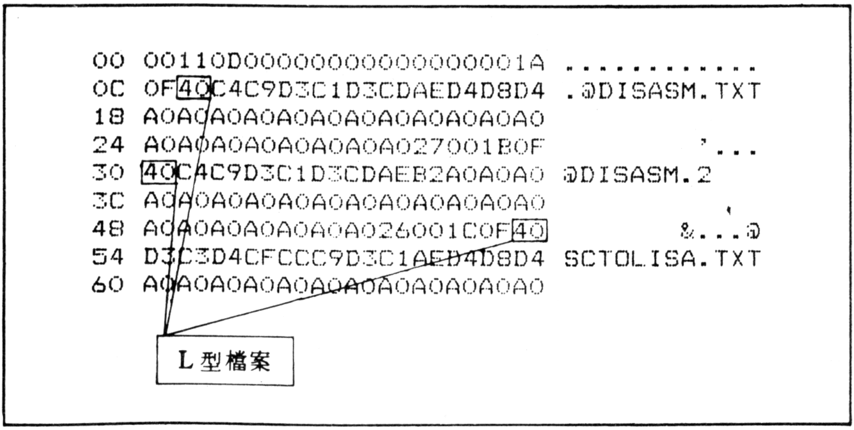


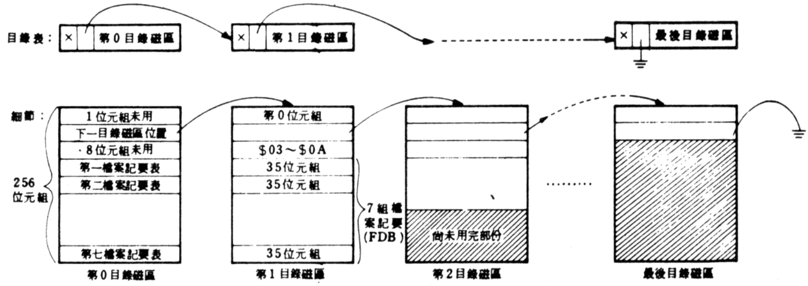
圖 2 − 7　LISA的 L 型檔案

第三欄：存放檔案的名稱，呼叫檔案時（ RUN, OPEN, EXEC 等）便是利用本欄來尋找的。由於 APPLE 的檔案名稱最多可用 30 個字母，因此本欄共有 30 格，若檔案名稱不足 30 格，則其後填入空白（ A0 ）。

第四欄：存放檔案佔用的長度，由於磁碟系統的磁片運作最小單位為磁區，因此本欄存的是磁區數目。

2－2 目錄表：CATALOG

那麼， FDB 又在哪裏呢？ FDB 是存在於目錄表裏頭，每個檔案有一個 FDB ，我們使用CATALOG這個指令來列表時，便是由這個目錄表當中的每個 FDB 第 3 欄（檔案名稱欄）將檔案名稱印出到螢幕上來。存放目錄表的磁區叫做目錄磁區，由於每個磁區只有 256 位元組，因此用作目錄表格的磁區每個最多只能存放 7 個檔案記要（ FDB ）*，正常情況， APPLE 共有 15 個目錄磁區，因此檔案最多 105 個。這些作為目錄表格的磁區，相互間是以 2 位元組的指位器來串聯起來的，每個目錄磁區第 1 、 2 兩個位元組便是用來作目錄磁區指位器，而最後一個目錄磁區的指位器內含值是（ 00 , 00 ），表示目錄至此結束。詳見下圖說明：



---

* 其餘做其他用途，後述。

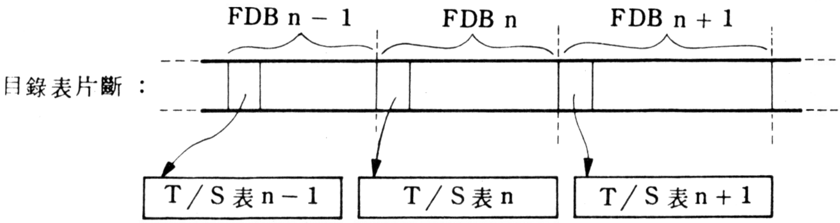


圖 2－8　目錄表、FDB、T／S表的相互關係

由上圖，我們就可以清楚的瞭解目錄表、 FDB 、T／S表之間的關係了。有關每個目錄磁區的詳細使用情形，請參閱下表：

目錄磁區使用摘要

位元組	用途
00－00	未使用
01－02	下一目錄磁區指位器
03－0A	未使用
0B－2D	第0號 FDB 共佔 35 位元組
2E－50	第1號 FDB 共佔 35 位元組
51－73	第2號 FDB 共佔 35 位元組
74－96	第3號 FDB 共佔 35 位元組
97－B9	第4號 FDB 共佔 35 位元組
BA－DC	第5號 FDB 共佔 35 位元組
DD－FF	第6號 FDB 共佔 35 位元組

現在，以 ZAP 來讀入實驗磁片的（ $11,$0F ）磁區，結果如下：

```
00 00110E0000000000000000013  ............
0C 0F04C8C5CCCCCFA0A0A0A0A0  ..HELLO
18 A0A0A0A0A0A0A0A0A0A0A0A0
24 A0A0A0A0A0A0A0A0830000000             ....
30 000000000000000000000000  ............
3C 000000000000000000000000  ............
48 000000000000000000000000  ............
54 000000000000000000000000  ............
60 000000000000000000000000  ............
```

檔案記要表 FDB

圖 2 － 9　目錄磁區

我們發現只有一個檔案記要表（FDB），由 $0B～$2D 共 35 位元組，其中（$13，$0F）便是我們剛才讀 HELLO T／S 表的磁區位置，因此 FDB 的前兩個位置便證實爲 T／S 表指位器了。接下來的 04 表示檔案爲 B 型檔（後述），再來便是 HELLO 檔案名稱（負 ASCII 值），雖然 HELLO 只有 5 個字，但是 FDB 共保留有 30 個位元組存放檔案名稱（這便是 DOS 手册說檔案名稱最長 30 個字元的原因了），因此其後塡入 25 個 A0 的空白 ASCII 碼。最後的（$83，$00）是長度位元組，低位元組在前高位元組在後，$83＝131 也就是 CATALOG 時顯示在 HELLO 檔名前面的那個數字，請注意，它包含了檔案 T／S 表及眞正資料所佔的所有磁區。

另外，圖 2 － 9 第 1、2 位元組的（$11，$0E）是下一個目錄磁區所在的位置，如果你繼續讀（$11，$0E）的內容，則會發現，目錄磁區指位置器的值爲（$11，$0D），依次下去一直到（$11，$01）磁區爲止，（$11，$01）磁區內的目錄磁區指位器爲（00，00）表示目錄磁區至此結束。這是目錄與 T

／S表不同之處，T／S表的長度依檔案長度而伸延，但目錄是在INIT時便準備好(Allocate)一串由($11,$0F)～($11,$01)的磁區專門給目錄使用。

除了圖2－9之外，繼續追踪目錄磁區，可以得到下列的結果：

```
00  00110D000000000000000000  ............
0C  000000000000000000000000  ............
18  000000000000000000000000  ............  (11, 0E)
24  000000000000000000000000  ............
30  000000000000000000000000  ............
```

↓

```
00  00110C000000000000000000  ............
0C  000000000000000000000000  ............
18  000000000000000000000000  ............  (11, 0D)
24  000000000000000000000000  ............
30  000000000000000000000000  ............
```

⋮

```
00  001101000000000000000000  ............
0C  000000000000000000000000  ............
18  000000000000000000000000  ............  (11, 02)
24  000000000000000000000000  ............
30  000000000000000000000000  ............
```

↓

```
00  000000000000000000000000  ............
0C  000000000000000000000000  ............  (11, 01)
18  000000000000000000000000  ............
24  000000000000000000000000  ............  目錄表最後
30  000000000000000000000000  ............  磁區
```

圖2－10　目錄的磁區空間

2－3 磁片內容表：VTOC

所以，我們都很清楚了，要找到某一檔案的資料，便要到目錄表由每個 FDB 的第 3 欄去尋找到擁有該檔案的名稱的 FDB，再由此 FDB 第 1 欄的T／S表指位器去找出T／S表位置，接著由T／S表上的資料磁區指位器便可找出眞正存資料的磁區了，正如第一章所說的一樣。

但是目錄表在哪裏呢？通常目錄的位置是如前一節我們所觀察到的由（$11,$0F）開始到（$11,$01）的 15 個磁區，不過目錄表眞正的位置是仍然由一個叫做磁片內容表＊（Volume Table Of Contents 簡稱VTOC）的表格來設定的，在VTOC上由第 1 、 2 位元組所構成的指位器，便是指向目錄表的起始位置，以 ZAP 來讀取（$11,$00）磁區，便可以看出VTOC的內容來：

＊早期電腦並不使用磁碟機而是使用磁帶（Reel），由於早期的磁帶是像盤式錄音帶一般的裝於紙盒內，因此稱一捲磁帶爲一Volume（IBM），後來就沿用於磁片，因此Volume應該是捲而不是册，此處Volume翻譯成磁片最爲適合。

```
00 00110F030000FE0000000000  ......^.....
0C 000000000000000000000000  ............
18 000000000000000000000000  ............
24 0000007A0000000000000000  ...Z........
30 1C0100002310000100000000  ....#.......
3C 0000000000000000FFFF0000  ........__..
48 FFFF0000FFFF0000FFFF0000  __..__..__..
54 FFFF0000FFFF0000FFFF0000  __..__..__..
60 FFFF0000FFFF0000FFFF0000  __..__..__..
6C FFFF0000FFFF0000FFFF0000  __..__..__..
78 FFFF00000000000FFFF0000   __......__..
84 000000003FFF000000000000  ....?_......
90 000000000000000000000000  ............
9C 000000000000000000000000  ............
A8 7FFF0000FFFF0000FFFF0000  __..__..__..
B4 FFFF0000FFFF0000FFFF0000  __..__..__..
C0 FFFF00000000000000000000  __..........
CC 000000000000000000000000  ............
D8 000000000000000000000000  ............
E4 000000000000000000000000  ............
F0 000000000000000000000000  ............
```

位元映射表

圖 2－11 VTOC顯示幕

位元映射表

VTOC的內容表示了什麼呢？它包含著有關於磁片整體內容的訊息，譬如：磁片版本、目錄表所在位置、磁片號碼（Volume Number）等等。其中有一項便是第一章提到的磁片空間位元映射表（Bit Map），這個表格管理磁片中每一磁區的使用情形。

位元映射的規則是這樣的：

磁軌上的每一磁區：	FEDC	BA98	7654	3210
對　應　的　位　元：	1111	1011	0110	1110
16　進　位　值：	F	B	6	E
	第 1 位元組		第 2 位元組	

圖 2 － 12　磁軌上磁區的位元映射

圖 2 － 12 當中的第 A 、 7 、 4 、 0 磁區已被使用，因此其位元映射值爲 FB 6E 。位元映射是依磁軌順序來排列，而由 VTOC 磁區的（ $38,$39 ）兩個位元組對應於第 0 號磁軌開始，每一磁軌之間空下兩個零位元組做間格（是否考慮以後當成半軌使用？），這便是我們由圖 2 － 11 的 VTOC 表中由位置 $38 ～ $C3 所看到的情形。

觀察圖 2 － 11 VTOC 的位元映射表，其情形是：由 0 － 1 節我們知道第 0 ～ 2 磁軌是由 DOS 佔用，因此圖 2 － 11 的（ $38 ,$39 ）、（ $3C,$3D ）、（ $40,$41 ）其值均爲 0（已使用），而接下來由對應於第 3 磁軌的（ $44,$45 ）開始，其位元值均爲 1 ，位元組值爲 FF 。而對應於 VTOC 及 CATALOG 的磁區是特別用途，當然不仍給一般檔案使用，因此第 17 （$11 ）磁軌的位元映射值（在 VTOC 的 $7C,$7D 位置）爲 0 。詳細計算 * 圖 1 － 11 當中由 $84 ～ $A8 的位元映射，共有 16 ＋ 2 ＋ 16 × 7 ＋ 1 ＝ 18 ＋ 112 ＋ 1 ＝ 131 磁區的位元映射值爲 0 ，這正

---

* 計算時請注意每磁區之間以兩個零位元組相間格，譬如 $ 62 , $ 63 便是。

好是我們在 2 － 0 節以 BSAVE 所造出來的 HELLO 程式佔用的空間。至此，我們對於位元映射已經清楚瞭解了。

每次要將資料寫入新磁區時， DOS 首先在VTOC的位元映射表上找出一個空的磁區（對應位元為 1 ），然後便將資料寫入，並且將其在位元映射表上的位元改為 0（表示使用中），而在檔案被消除（DELETE）時， DOS 便將位元映射表上對應於此檔案的位元重置為 1，如此其它檔案才能夠重新來使用這些磁區。

除了目錄指位器、位元映射表之外 VTOC 的其它位元組做什麼用呢？請見下表說明：

VTOC磁區使用摘要

位元組	用途
00 － 00	未使用
01 － 02	指向 CATALOG 第一磁區位置的指位器
03 － 03	DOS 版本（現為 03 ，即 VER 3.3 ）
04 － 05	未使用
06 － 06	磁片號碼（ Volume Number ）
07 － 26	未使用
27 － 27	T／S 表所能容納的資料磁區指位器數目（$7A）
28 － 2F	未使用
30 － 30	最後一次所寫之磁軌號碼
31 － 31	改換磁軌時，每次變換之磁軌數目（向前或向後一軌）
32 － 33	未使用
34 － 34	磁片的磁軌數目
35 － 35	每一磁軌上的磁區數目
36 － 37	每一磁區的位元組數目（低位元在前）
38 － C3	位元映射表
C4 － FF	未使用

至此，整個磁片的檔案構造已說明完畢， APPLE 要讀一個檔案資料時，首先至VTOC找出CATALOG位置，然後由CATALOG找出檔案的T／S表，再由T／S表去找到眞正的資料。而在寫資料時，也是依上述的方式去找到資料磁區，如果寫滿該磁區而欲繼續寫入時，便由VTOC的位元映射表找出一個未用的磁區，將對應位元改爲0，在T／S表造出一組指位器指向此新磁區（如果T／S磁區已滿，則要先連結到新T／S磁區），然後才可再寫入資料。

整個磁片檔案系統詳見下圖：

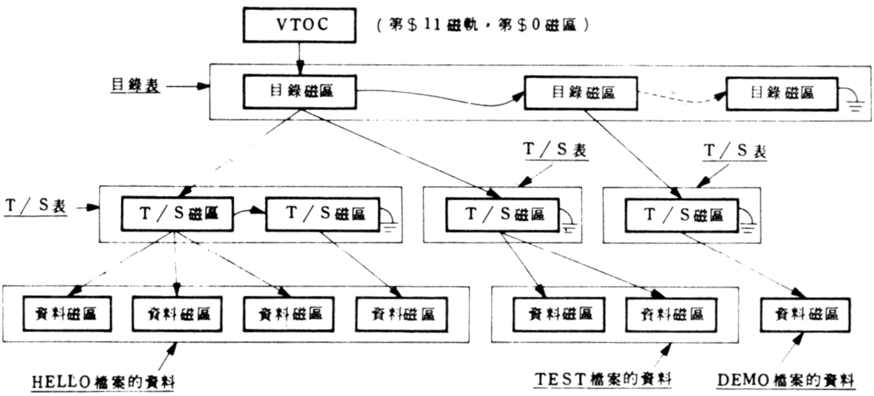


圖2－13　磁片檔案系統圖

2－4 檔案類型

對於檔案系統，最後還有一點未予瞭解的，那就是：各型檔案的存放規則。譬如說，我們在圖2－2所看到的HELLO程式是Applesoft檔案，其最前面的位元組存放的是程式總長度，但

是文字檔（T檔）卻是一開始就是資料而沒有長度位元組。到底各種檔案的規則是怎樣的呢？

為了試驗，首先我們在實驗磁片上做出各類檔案：

1.

鍵入底下程式，以TEST爲檔名SAVE到磁片上，然後RUN

```
10 D$ = CHR$ (4)
20 PRINT D$"OPEN TEXT"
30 PRINT D$"WRITE TEXT"
40 PRINT "THIS IS A TEST"
50 PRINT "THIS IS 123+ABC"
60 PRINT "THIS IS END"
70 PRINT D$"CLOSE"
```

執行成功後，鍵入LOCK TEST

本程式執行後爲造出一個叫TEXT的文字檔

2.

將上面程式修改如下，然後RUN

```
10 D$ = CHR$ (4)
20 PRINT D$"OPEN TEXT.1"
30 PRINT D$"WRITE TEXT.1"
70 PRINT D$"CLOSE"
```

本程式執行後會造出一個叫TEXT.1的文字檔，其內容爲空白

3.

請用一片DOS MASTER 磁片叫入整數BASIC，換上實驗磁片，鍵入本程式，以TEST.I 為檔名 SAVE 到磁片上，然後RUN

```
>LIST
  10 D$="": REM  CONTROL-D
  20 PRINT D$;"OPEN TEXT.I"
  30 PRINT D$;"WRITE TEXT.I"
  40 PRINT "THIS IS A TEST"
  50 PRINT "THIS IS 123+ABC"
  60 PRINT "THIS IS END"
  70 PRINT D$;"CLOSE"
  80 END
```

本程式執行後會造出一個叫 TEXT.I 的文字檔，其內容與TEXT相同

4.

CATALOG 之後，在實驗磁片上的檔案應該是：

```
]CATALOG

DISK VOLUME 254

 B 131 HELLO
*A 002 TEST
 T 002 TEXT
 T 001 TEXT.1
 I 002 TEST.I
 T 002 TEXT.I
```

現在便可以開始來觀察實驗磁片內各型檔案的內容了。

首先，要讀VTOC的內容以找出目錄表位置，以正常 DOS 來說，其此時VTOC第1、2位元組的值是$11,$0F，因此我們便到 $11 磁軌內的 $0F 磁區來找目錄表的內容，結果如下:

```
00 00110E000000000000000013 ............
0C 0F04C8C5CCCCCFA0A0A0A0A0 ..HELLO
18 A0A0A0A0A0A0A0A0A0A0A0A0
24 A0A0A0A0A0A0A083001C0F       ....
30 82D4C5D3D4A0A0A0A0A0A0A0 .TEST
3C A0A0A0A0A0A0A0A0A0A0A0A0
48 A0A0A0A0A0A0A002001D0F00      .....
54 D4C5D8D4A0A0A0A0A0A0A0A0 TEXT
60 A0A0A0A0A0A0A0A0A0A0A0A0
6C A0A0A0A0A0A002001E0F00D4      .....T
78 C5D8D4AEB1A0A0A0A0A0A0A0 EXT.1
84 A0A0A0A0A0A0A0A0A0A0A0A0
90 A0A0A0A0A001001F0F01D4C5     .....TE
9C D3D4AEC9A0A0A0A0A0A0A0A0 ST.I
A8 A0A0A0A0A0A0A0A0A0A0A0A0
B4 A0A0A0A00200200F00D4C5D8    .. ..TEX
C0 D4AEC9A0A0A0A0A0A0A0A0A0 T.I
CC A0A0A0A0A0A0A0A0A0A0A0A0
D8 A0A0A0020000000000000000   .........
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
FC 00000000                 ....
```

圖 2－14　目錄表的內容

＊由於程式鍵入過程，可能有多次修改及SAVE，結果可能與本圖稍有不同（尤其是T／S表位置），不過大致應該相同。

結果目錄表的內容正是剛才我們CATALOG後在螢幕上看到的內容。首先是HELLO程式，其T／S表由第（$13,$0F）磁軌開始，類型是B型未上鎖（04，若上鎖則為84），長度為$83＝131磁區。接下來的檔案是TEST程式，T／S表在（$1C,$0F），類型是A型已上鎖（82），長度為兩個磁區。TEXT的T／S表在（$1D，0F），類型是文字檔未上鎖（00），長度為兩個磁區。TEXT.1的T／S表在（$1E，0F），類型是文字檔未上鎖（00），長度為一個磁區。TEST.I的T／S表在（$1F，$0F），類型是I型檔（01），長度為兩個磁區。最後是TEXT.I，其T／S表是在（$20，0F），類型為文字檔（00），長度為兩個磁區。

根據這些T／S表指位器，我們可以讀出各T／S表的內容，而最後由T／S表的資料磁區指位器追踪到資料磁區位置而讀出其眞正的資料內容來：

B型檔：HELLO

依照目錄順序，首先讀到的是HELLO的內容，如下圖：

程式起始位址　程式長度

```
00 0008FF7F000000C942000000 ..__...IB...
0C 0000004A4A09C0853FA95C85 ...JJ.@.?)\.
18 3E18ADFE086DFF088DFE08AE >.-^.M_..^..
24 FF083015BD4D08853DCEFF08 _.0.=M..=N_.
30 ADFE088527CEFE08A62B6C3E -^..'N^.&+L>
3C 00EEFE08EEFE082089FE2093 .N^.N^. .^ .
48 FE202FFBA62B6CFD08000D0B ^ /[&+L]....
54 0907050301 0E0C0A08060402 ............
60 0F002064A7B008A900A88D5D .. D'0.).(.]
6C B69140ADC5B54CD2A6AD5DB6 6.@-E5LR&-]6
78 F008EEBDB5D003FEBEB5A900 P.N=5P.N>5).
84 8D5DB64C46A58DBCB520A8A6 .]6LF%.<5 (&
```

圖 2－15　B 型檔的內容

圖中，最初兩個位元是 BSAVE 時的起始位址，我們用的是 A$800，因此其值為（00，08），接下來是程式長度，我們用的是 L$7FFF，因此其值為（FF，7F），再來就是程式內容了。由於 BSAVE 所存的是記憶體內含值，而由 $800 開始存的是 BASIC 程式，雖然我們已用 NEW 來清除程式，但其實只是將最初 3 個位元組設為 0 而已*，其它內容仍舊是存在的，因此就有許多看似雜亂無章的內容了。

A型檔：TEST

接下來的是 Applesoft 的程式 TEST：

---

* 見第一冊

```
程式長度   原本存在＄801的值

00 84000D080A004424D0E72834  ......D$PG(4
0C 29002008140OBA4424224F50  ).  ...:D$"OP
18 454E2054455854220034081E  EN TEXT".4..
24 00BA44242257524954452054  .:D$"WRITE T
30 45585422004A082800BA2254  EXT".J.(.:"T
3C 484953204953204120544553  HIS`IS A TES
48 5422006108320OBA22544849  T".A.2.:"THI
54 53204953203132332B414243  S IS 123+ABC
60 220074083C00BA2254484953  ".T.<.:"THIS
6C 2049532045 4E4422008308 46   IS END"...F
78 00BA442422434C4F53452200  .:D$"CLOSE".
84 000044452200000026000000  ..DE"...&...
90 000000000000000000000000  ............
9C 000000000000000000000000  ............

3個零位元組表示程式到此結束
長度＝85＋1－2＝84（HEX）
```

其實，A型檔與前面的B型檔其資料存放格式是一樣的，只是BASIC 程式的起始位址是固定的，因此不用再記入磁片檔案內，這是與B型檔需要起始位址不同之處。

文字檔：TEXT

```
00 D4C8C9D3A0C9D3A0C1A0D4C5  THIS IS A TE
0C D3D48DD4C8C9D3A0C9D3A0B1  ST.THIS IS 1
18 B2B3ABC1C2C38DD4C8C9D3A0  23+ABC.THIS
24 C9D3A0C5CEC48D0000000000  IS END......
30 000000000000000000000000  ............
3C 000000000000000000000000  ............
```

文字檔有兩種，此處所使用的是循序檔，循序文字檔一開始就是資料，沒有起始位址也沒有長度位元組。本檔案的內容是以TEST這個程式的第40、50、60行的PRINT 寫進來的，PRINT 指令若後邊未加逗號或分號，則會自動產生一個RETURN 碼（ 8D ），通常文字檔便是使用這個RETURN 碼來做爲每筆資料之間的分隔，圖中＄0E 、＄1E 位置上的 8D 便是用來分隔THIS IS A TESE 、THIS IS 123＋ABC 及THIS IS END 這3筆資料的。

文字檔：TEXT・1

```
00  000000000000000000000000    ............
0C  000000000000000000000000    ............
18  000000000000000000000000    ............
24  000000000000000000000000    ............
30  000000000000000000000000    ............
```

＊這是TEXT.1的T／S磁區，不是資料磁區

這個文字檔是我們將TEST程式中的第40、50、60行去掉而造成的，由於沒有寫入（ PRINT ）資料，因此根本沒有資料磁區，T／S表全部空白，此處印出來的是T／S磁區而不是資料磁區。由於TEXT.1這個檔案只使用了一個T／S磁區，因此其長度爲1 ，這可由CATALOG螢幕上看出。其他種檔案，雖然內容爲0 ，但是至少也要記錄長度或起始位址，因此必定要使用到資料磁區，這是爲何文字檔的最小

長度爲 1 磁區，而其它檔案爲 2 磁區的原因。

I 型檔：TEST・I

程式長度

```
00 9200160A00C44070288842903 .....D@F(.).
0C 5DA0C3CFCED4D2CFCCADC401  ] CONTROL-D.
18 15140061C4404528CFD0C5CE  ...AD@E(OPEN
24 A0D4C5D8D4AEC92901161E00   TEXT.I)....
30 61C4404528D7D2C9D4C5A0D4  AD@E(WRITE T
3C C5D8D4AEC92901152800612 8 EXT.I)..(.A(
48 D4C8C9D3A0C9D3A0C1A0D4C5  THIS IS A TE
54 D3D42901163200612 8D4C8C9 ST)..2.A(THI
60 D3A0C9D3A0B1B2B3ABC1C2C3  S IS 123+ABC
6C 2901123C00612 8D4C8C9D3A0 )..<.A(THIS
78 C9D3A0C5CEC429010F460061  IS END)..F.A
84 C4404528C3CCCFD3C5290105  D@E(CLOSE)..
90 5000510192C529019600000 0 P.Q..E).....
9C 000000000000000000000000  ............
A8 000000000000000000000000  ............
B4 000000000000000000000000  ............
```

程式結束，51爲END的指令代碼，01爲該行結束指示（END OF LINE）

長度＝93＋1－2＝92（HEX）

整數BASIC 檔案的格式與APPLESOFT 相同，其差異除了BASIC 程式格式不同之外，就只有在 FDB 的類型位元組了，請試著將TEST.I的 FDB 中類型位元組改成 02，CATALOG之後就變成A 型檔了，LOAD再LIST ，結果如何呢？

```
]CATALOG

 B 131 HELLO
*A 002 TEST
 T 002 TEXT
 T 001 TEXT.1
 A 002 TEST.I
 T 002 TEXT.I
```

更改 TEST．I 的類型位元組後，CATALOG的情形

文字檔：TEXT．I

最後是TEXT．I 這個由整數 BASIC 程式TEST．I 所造成的文字檔：

```
00 D4C8C9D3A0C9D3A0C1A0D4C5 THIS IS A TE
0C D3D48DD4C8C9D3A0C9D3A0B1 ST.THIS IS 1
18 B2B3ABC1C2C38DD4C8C9D3A0 23+ABC.THIS
24 C9D3A0C5CEC48D0000000000 IS END......
30 000000000000000000000000 ............
3C 000000000000000000000000 ............
```

結果與由APPLESOFT 程式TEST所造成的TEXT無論其格式或類型完全相同，這便是APPLE DOS 手册第七章中所說的能夠以文字檔截取程式，而將整數 BASIC 轉換成APPLESOFT程式的原因了，因爲二者的文字檔是完全相同的。至於循序檔與隨機檔其存放情形也是完全相同的，只是DOS（ FMS ）處理的差異而已。

參考文獻：

Ref 1 APPLE Ⅱ　DOS 使用手册附錄C

本章提要

花了很多工夫瞭解了磁片構造之後，這些知識是否實際而能派上用場呢？本章告訴您：

①I／O ERROR怎麼辦？

②快I／O ERROR了！怎麼知道？

③磁片滿了嗎！

④DOS 當了！在 RAM 裡頭的程式如何保存？

3 I/O ERROR

或許我們可以不管 APPLE 的磁片是如何又如何安排的，怎樣又怎樣組織。直到有一天，I／O ERROR………

3－0 修理磁片的準備工作

對一個眞正的電腦用家而言，其專用來存放資料的磁片，往往比程式磁片還要多，程式磁片損壞了仍可以花錢購買得到，但存放著僅此一家資料的磁片，可眞是損壞不得！或許您會認爲多拷貝一份備用不就可以了嗎？可是對於擁有數百張（很快就上百張了！）磁片的人，備份磁片可也眞是麻煩呢！所以總之，會有那麼一天，您珍貴的磁片壞了，而又沒有備份………

對於這種情形，我們所學到的 DOS 知識便可派上用場了。

磁片的故障基本上可區分為兩大類型：

①資料欄故障：此類故障較易修護，大致上以 ZAP 、FID 及 APPLE 的MASTER CREATE相互配合便可修好。其細節於下節說明。

②位置欄故障：位置欄的故障只有 nibble editor之類的程式才能直接修護。但是對於故障區域較大或不熟悉 nibble editor 的使用者，直接修護往往不甚方便也不太安全。較好的處理方法是：

位 置 欄 故 障 的 處 置 法
由於位置欄已故障，只有用 INIT來修護。但 INIT不能直接用於原磁片，否則全部資料會受到破壞，因此： 1. INIT一張新磁片。 2. 以可以限定磁軌的 COPY程式，如COPY Ⅱ PLUS的BIT COPY 或 CRAZY COPY來將未故障的磁軌 COPY至新磁片上。 3. 以 ZAP 或其他可以限定磁區的COPY 程式來將故障磁區所在磁軌上的其它完整磁區寫至新磁片上。 如此一來，新磁片上便含有所有未損壞的磁區資料，並且所有位置欄均是完整無缺，接下來只要以資料欄故障的方法來處理便可以了。

表 3 － 0

故障位置的檢查

要找出磁片的故障位置，有時並不簡單，尤其是對於資料磁區故障的檢查，往往由VTOC一直查到故障所在就要讀取磁區不下十數回。如果我們能夠先檢查出哪個磁區故障，由磁區位置座標就很容易判斷故障種類而不必從頭找起了。

故障磁區位置	故障種類
0～2號磁軌	DOS
($11，$00)	VTOC
$11，$01～$0F	目錄表
其他	T／S表或資料磁區

表3－1

底下就是一個磁區檢查程式，鍵入無誤 ASM 後，請BSAVE SECTOR EXAM，A$5000，L$283。本程式會檢查每一磁區，並將故障號碼顯示於螢幕上：

```
!L    1 ;
    2 ********************************
    3 *   SECTOR SWEEPER SWEEPER    *
    4 ********************************
    5 GP          EPZ  $0006
    6 GPL         EPZ  GP
    7 GPH         EPZ  GP+1
    8 PREG        EPZ  $48
    9 LOCPR       EQU  $03E3
```

```
RWTS       EQU $03D9
IOB        EQU $B7E8
IBSLOT     EQU IOB+1     ┐
IBDRVN     EQU IOB+2     │
IBVOL      EQU IOB+3     │
IBTRCK     EQU IOB+4     │
IBSCTR     EQU IOB+5     ├ IOB位置
IBBUFL     EQU IOB+8     │
IBBUFH     EQU IOB+9     │
IBCMND     EQU IOB+12    │
IOBERR     EQU IOB+13    ┘
COUT       EQU $FDED
CROUT      EQU $FD8E
PRBYTE     EQU $FDDA
SCINDOS    EQU 16              ; DOS 3.3
*
*
*----BEGIN.PROGRAM----------
*
           ORG $5000
           JMP GO
*
*----DATA.AND.BUFFERS-------
*
SECLIST    BYT $0,$6,$C,$2,$8,$E
           BYT $4,$A,$1,$7,$D,$3
           BYT $9,$F,$5,$B
SECINDX    NOP
OLDIOB     DFS $11
BUFLS2     ADR BUFFER
BUFFER     DFS $100
MSGADDR    ADR MSG0            ; 字串位址
           ADR MSG1
           ADR MSG2
           ADR MSG3
MSG0       ASC "INSERT DISK;  THEN HIT RETURN "
           ASC "TO CHECK, OTHER TO END.^"
MSG1       ASC "ERROR ^"
```

```
MSG2       ASC " FOUND FOR TRACK ^ "
MSG3       ASC ", SECTOR ^"
*
*-----BEGIN.SUBROUTINES--------
*
*--BENERAL-PURPOSE.DISK.ROUTINE-
WDISK      LDA #$02            ; Write
           BNE STACMND
RDISK      LDA #$01            ; read
STACMND    STA IBCMND
           LDA BUFLS2          ; 設定暫存區指位器
           STA IBBUFL
           LDA BUFLS2+1
           STA IBBUFH
           LDA IBTRCK          ; 合法的磁軌？
           BMI PARMERR
           CMP #35
           BPL PARMERR
           LDA IBSCTR          ; 合法的磁區？
           BMI PARMERR
           CMP #SCINDOS
           BPL PARMERR
           LDA #$00            ; 任何VOL＃均可
           STA IBVOL
           JSR LOCPR
           CLC
           JSR RWTS            ; 讀取
           LDA #$00
           STA PREG
           BCS OOPS
           RTS
OOPS       LDA IOBERR          ; return時ERR＃→A
           RTS
PARMERR    LDA #$00            ; 不合法的T／S表
           SEC
           RTS
*------------------------------
*--PRINT.MESSAGE.#N,.N=ACC
```

```
*--PRRMSG.PRINTS.A.RETURN.FRIST,
*--PRMSG.DOESN'T.
PRRMSG    PHA                    ;印字串的副程式
          JSR  CROUT
          PLA
PRMSG     ASL
          TAY
          LDA  MSGADDR,Y
          STA  GPL
          INY
          LDA  MSGADDR,Y
          STA  GPH
          LDX  #$00
LDAGPX    LDA  (GP,X)
          CMP  #"^"              ;字串結束?
          BEQ  RTN2
          JSR  COUT
          INC  GPL
          BNE  LDAGPX
          INC  GPH
          JMP  LDAGPX
RTN2      RTS
*--------------------------------------------
*-----BEGIN.MAIN.PROGRAM-----     ;主程式由此開始
*
GO        PHP                    ;存P,A,Y
          PHA
          TYA
          PHA
          LDX  #$10
IOBSAVE   LDA  IOB,X             ;存目前DOS使用的IOB
          STA  OLDIOB,X
          DEX
          BPL  IOBSAVE
GO1       LDA  #$00              ;由0號磁軌開始
          STA  IBTRCK
          STA  SECINDX           ;由0號磁區開始
          LDA  #$00
```

```
         JSR PRRMSG        ;要求按 RETURN
         LDA $C010
CHKAGIN  LDA $C000
         BPL CHKAGIN
         CMP #$8D
         BEQ DOOPRS1
         LDX #$10
IOBRSTO  LDA OLDIOB,X      ;若不是 RETURN 則存
         STA IOB,X         ;回DOS原來的IOB
         DEX
         BPL IOBRSTO
         LDA $C010
         LDX #$00
         LDA #$8D
         STA $200,X
         PLA               ;取回A,Y,P
         TAY
         PLA
         PLP
         JMP $E003         ;回到BASIC結束
DOOPRS1  JSR CROUT
DOOP     LDY SECINDX
         LDA SECLIST,Y     ;索引各磁區
         STA IBSCTR
         JSR RDISK         ;讀
         BCC NOFAULT       ;故障?
         PHA
         LDA #$01
         JSR PRRMSG        ;印出故障號碼
         PLA
         JSR PRBYTE
         LDA #$02
         JSR PRMSG
         LDA IBTRCK
         JSR PRBYTE
         LDA #$03
         JSR PRMSG
         LDA IBSCTR
         JSR PRBYTE
```

```
NOFAULT   INC SECINDX        ;下一磁區
          LDA SECINDX
          CMP #SCINDOS       ;完成--磁軌？
          BNE DOOP
          LDA #$00
          STA SECINDX
          INC IBTRCK
          LDA IBTRCK
          CMP #35            ;全部OK？
          BNE DOOP
          JMP GO1
          END
```

程式3－0　故障磁區檢查程式

```
*5000.5013

5000- 4C F2 51 00 06 0C 02 08
5008- 0E 04 0A 01 07 0D 03 09
5010- 0F 05 0B 00
*5025.5026

5025- 27 50
*5127.5282

5127- 2F
5128- 51 65 51 6C 51 7F 51 C9
5130- CE D3 C5 D2 D4 A0 C4 C9
5138- D3 CB BB A0 A0 D4 C8 C5
5140- CE A0 C8 C9 D4 A0 D2 C5
5148- D4 D5 D2 CE A0 D4 CF A0
5150- C3 C8 C5 C3 CB AC A0 CF
5158- D4 C8 C5 D2 A0 D4 CF A0
5160- C5 CE C4 AE DE C5 D2 D2
5168- CF D2 A0 DE A0 C6 CF D5
5170- CE C4 A0 C6 CF D2 A0 D4
5178- D2 C1 C3 CB A0 DE A0 AC
5180- A0 D3 C5 C3 D4 CF D2 A0
5188- DE A9 02 D0 02 A9 01 8D
5190- F4 B7 AD 25 50 8D F0 B7
5198- AD 26 50 8D F1 B7 AD EC
51A0- B7 30 24 C9 23 10 20 AD
51A8- ED B7 30 1B C9 10 10 17
51B0- A9 00 8D EB B7 20 E3 03
51B8- 18 20 D9 03 A9 00 85 48
51C0- B0 01 60 AD F5 B7 60 A9
51C8- 00 38 60 48 20 8E FD 68
51D0- 0A A8 B9 27 51 85 06 C8
51D8- B9 27 51 85 07 A2 00 A1
51E0- 06 C9 DE F0 0C 20 ED FD
51E8- E6 06 D0 F3 E6 07 4C DF
51F0- 51 60 08 48 98 48 A2 10
51F8- BD E8 B7 9D 14 50 CA 10
5200- F7 A9 00 8D EC B7 8D 13
5208- 50 A9 00 20 CB 51 AD 10
5210- C0 AD 00 C0 10 FB C9 8D
5218- F0 1C A2 10 BD 14 50 9D
5220- E8 B7 CA 10 F7 AD 10 C0
5228- A2 00 A9 8D 9D 00 02 68
5230- A8 68 28 4C 03 E0 20 8E
5238- FD AC 13 50 B9 03 50 8D
5240- ED B7 20 8D 51 90 20 48
5248- A9 01 20 CB 51 68 20 DA
5250- FD A9 02 20 D0 51 AD EC
5258- B7 20 DA FD A9 03 20 D0
5260- 51 AD ED B7 20 DA FD EE
5268- 13 50 AD 13 50 C9 10 D0
5270- C8 A9 00 8D 13 50 EE EC
5278- B7 AD EC B7 C9 23 D0 B9
5280- 4C 01 52
*
```

程式3－0的機械碼

本程式之構造很簡單，其主要工作是由0號磁軌開始讀取每一磁區，若無法讀取便印出錯誤訊息 40 以及磁軌磁區號碼。磁區的讀取主要是利用 DOS 的RWTS及其 IOB ，詳情請參閱第4及第6章，至於其他如印字副程式等參考程式附註當可明瞭。如果您鍵入的是機械碼，則請注意前頭的一些位址是不連續的。

3－1 故障磁區的檢查

由程式3－0我們可以找出故障磁區的號碼，並且由表3－1可以判斷出該磁區的種類。如果是屬於位置欄的故障*，以表3－0的方法就可以將之改成資料欄故障的情況加以處理。因此，我們就來看看資料欄故障的修護方法：

VTOC故障

VTOC含有整體磁片的重要訊息，譬如第1、2位元組故障會使得DOS 無法找到目錄表起始磁區的位置，而位元映射表的故障會使新檔案覆蓋原存在的檔案。解救的辦法是，重造一個合法的VTOC。利用 ZAP 程式，清除ZAP的BUFFER，只在第1、2位元組填入（$11，$0F），然後寫入（$11，$00 ）磁區，這樣便造出一個可用但卻空白（沒有位元映射表等資料）的VTOC。現在故障磁片已可以用CATALOG列表，也可以用FID COPY 了，但是因為沒有位元映射表，因此

---

*凡是以資料欄故障方式無法處理（不能寫入）者，就是位置欄故障。

絕對不可以寫（SAVE）入任何資料。INIT 一張新磁片，用 FID 將檔案COPY 過來，FID 會自動造出正確的 VTOC，磁片便得救了。

如果發現 ZAP 無法將BUFFER 的資料寫入（$11，$00）磁區，則表示爲位置欄故障，這種情形以表 3 — 0的方法來處理便可以了。處理後，不必再以 ZAP 來寫入VTOC，因爲 INIT 已在新磁片上建好VTOC，這時只要用 FID 來COPY 所有檔案至另一張新INIT 的磁片便可以了。

CATALOG故障

如果程式 3 — 0 所檢出的故障位置在目錄磁區，則問題就較複雜。因爲 APPLE 的目錄與T／S表之間只用一個指位器做單向連結，目錄磁區一損壞，就難以找回T／S表了。幸好在 Bag of Trick 這張磁片上有一個叫 FIXCAT 的程式，此程式專門用來修理 VTOC或目錄磁區的故障。其方法是掃描磁片，找出可能是離散的T／S表，在目錄表上建立 FDB，最後由使用者來將額外建立出來的不正確檔案消去。

FIXCAT 很容易使用，BRUN 之後會訊問許多問題，第一個是 TIMEOUT：

```
AUTOMATIC TIMEOUT IN SECONDS?
(0 TO 255 SECONDS, 0 MEANS NO TIMEOUT)
 0
```

如果您每次均要親自回答問題，則可以回答0，如此 FIXGAT 每個問題均要等候您的回答才會再進行下一步驟。如果您要讓程式自動進行，可以回答“1”(或2)，如此 FIXCAT 每次詢問若等1(或2)秒鐘不見回答就逕行執行下一步驟了。

FIXCAT 若訊問：

```
READ EXISTING CATALOG FROM DISKETTE
OR START FROM SCRATCH? ("R" OR "S")
 R
```

要回答'R'，否則 FIXCAT 會將完整的檔案也當成故障檔案處理。當 FIXCAT 訊問：

```
SCAN FOR LOST OR DELETED FILES?
 Y
```

這時要答Y，FIXCAT 便會搜尋與目錄離散的T／S表，並且顯示其開頭的資料磁區內容(即程式或資料的內容)，依此你便可以判斷是否為要找回的檔案或無用的檔案。如果您有在程式開頭加標題的習慣，此時一定會記得檔案名稱的，FIXCAT 會依你所給的檔名來修護此檔案。若你無法給出檔名，FIXCAT 便以UNKNOW FILE #1、2、3

、……的方式來命名，等程式結束後你可一一載入這些UNKNOWN程式以判別其檔名。對於某些多次DELETE、SAVE的程式，其T／S表已廢棄不用，但可能尚未被其他檔案覆蓋，FIXCAT 會造出許多類似的檔案來，此時唯有一一叫出判斷一途了。如果您有在檔名加上版本號碼（FILE.3.2）的習慣，這時就很有幫助了。

修護檔案時，FIXCAT 會要求您指定檔案類型，對於T、I、A、B、R等型式 FIXCAT大多等自動判別出來。但是LISA的L檔則會被當成B檔，所以修護完畢後，必須再用 ZAP 將其類型位元組（檔名的前一個）由 04 改為 40 才能由LISA載入。

T/S表及資料磁區的故障

如果目錄磁區完整無缺，由 FDB 便可以找到T／S表的所在。如果T／S表發生故障，復原就很困難了，唯一的辦法是使用nibble editor來分析其殘餘資料。但是，用nibble editor 讀到的資料是未解碼的情形，您必須以第0章所介紹的6／2編碼法來將它解回原來的樣子，這時的T／S表內容由於已損壞，因此得小心檢查，將明顯的故障標出來。T／S表內最重要的資料是資料磁區指位器，記下這些指位器的值，然後用ZAP讀取每一個指位器所指的磁區內容，如果發現某些內容怪異或前後不連貫的磁區，則可能是指位器錯亂了。將所有正確的資料磁

區整理出來後，以 ZAP 來建立一個T／S表，然後 COPY 到另外磁片上進行檔案（或程式）修整便可以了。

當然上述的手續是瑣碎而複雜的，只在不得已的情況使用，如果您保有原檔案（或程式）的原始資料，最好還是再鍵入一遍，或許更迅速呢！至於資料磁區的故障當然也可以仿T／S磁區的方式來處理，但是只有256位元組的資料磁區是否也值得花這種工夫呢？通常的方法是將該磁區的原指位器消去（請注意指位器必須是連續的），如果故障的程式是T檔則用文件處理程式（Word processor ）來加以修整；如果故障的是BASIC （或R、L、B檔）則依其所屬的編輯方式加以修理。

DOS程式的故障

DOS 是存在於0～2號磁軌上，由於每次起動均由 DOS 磁軌開始，如果磁碟機故障 DOS 是首先遭到破壞的部份。有時候誤用 INIT 指令，若及早 RESET ，則破壞的也可能只是 DOS 磁軌而已。請以我們的實驗磁片， INIT A ，聽到 INIT第0磁軌後，馬上 RESET ，這時 PR# 6已不能起動 DOS 了，使用 COPY Ⅱ PLUS 的BIT COPY 或其他可以指定磁軌的COPY程式，將一張完整磁片的0至2磁軌COPY到實驗磁片上來，再次 PR # 6 一切便完好如初了。如果磁區位置欄沒有損壞的話，以APPLE MASTER CREATE 程式來重寫 DOS ，也可以將DOS 修好。

3－2 I/O ERROR的預防措施

當一片新磁片開始使用後，其性能便開始受到塵埃、磁場、自然衰退、意外損傷等的影響，經過一段時間之後便逐漸退化了。但是哪一張磁片退化至什麼程度呢？使用者是無從知道的，除非是I／O ERROR的那一刻………

在電腦設計裏，有一個專題叫做意外容忍（Fault tolerance），這是說在硬體或軟體上，如果發生內在故障或意外環境變化時（設計時未能預測的環境），電腦最好能夠承受而不停止其正常運作。譬如APPLE在讀取資料時，可能因瞬間的磁碟機運轉不穩或電訊干擾而無法讀取，DOS對於這種情形的意外容忍方式是：重複嘗試讀48次，若仍無法讀取，則令磁頭歸零以保位置正確，再重新至該磁區嘗試48次，若總共96次均告失敗，DOS才放棄努力而宣告I／O ERROR。

利用DOS的這種特性，我可由DOS的嘗試次數來判斷磁片的狀況，對於超過16次（$10）的磁片，其性能已有問題，而超過48次（會聽到磁頭歸零的振動聲）的磁片則一定要拷貝到新片了。本程式計數最多為$30（48），因此聽磁頭振動聲時，要自行加上$30。

```
2 ************************************
3 *    CHECK ERROR CONDITION ....     *
4 ************************************
5 *
6          ORG $300
7 *
```

DOS修改

```
         LDA #$20          ; JSR CHECK
         STA $BDC9
         LDA #CHECK
         STA $BDCA
         LDA /CHECK
         STA $BDCB
         LDA #RECAL        ; JSR RECAL
         STA $BE6E
         LDA /RECAL
         STA $BE6F
         LDA #$01
         STA $22
         LDX #$00
         STX TEMP          ; TEMP ← 0
HEADER   LDA TABLE,X       ; 印標題
         STA $400,X
         INX
         CPX #40
         BNE HEADER
         RTS
TABLE    ASC "DATA RETRIES"
         ASC "=            PRESENT  "
         ASC "TRACK      "
*
CHECK    LDA #$30
         SEC
         SBC $578
         CMP TEMP
         BLT RETURN1
         STA TEMP
         PHA               ; 印字開始
         LSR               ; 高半位元組
         LSR
         LSR
         LSR
         ORA #$B0          ; 換成ASCII碼
         CMP #$BA
         BCC PRINT
         ADC #$06
```

```
PRINT     STA  $40D         ；印出來
          PLA
          AND  #$0F         ；取低半位元組
          ORA  #$B0
          CMP  #$BA
          BCC  PRINTL
          ADC  #$06
PRINTL    STA  $40E         ；印出來
*
RETURN1   DEC  $578         ；原程式
          RTS
* 印出磁軌號碼
RECAL     PHA
          LSR               ；除以 2 ＝磁軌號碼
          PHA
          LSR
          LSR
          LSR
          LSR
          ORA  #$B0
          CMP  #$BA
          BCC  PRINT1
          ADC  #$06
PRINT1    STA  $424
          PLA
          AND  #$0F
          ORA  #$B0
          CMP  #$BA
          BCC  PRINL1
          ADC  #$06
PRINL1    STA  $425
          PLA
          JMP  $BE8E        ；到原程式
TEMP      DFS  1
          END
```

＊本程式由 CALL - APPLE All about DOS 改良而成。

程式 3－1　磁片退化檢查程式

```
*300.3A9

0300-  A9  20  8D  C9  BD  A9  55  8D
0308-  CA  BD  A9  03  8D  CB  BD  A9
0310-  85  8D  6E  BE  A9  03  8D  6F
0318-  BE  A9  01  85  22  A2  00  8E
0320-  A9  03  BD  2E  03  9D  00  04
0328-  E8  E0  28  D0  F5  60  C4  C1
0330-  D4  C1  A0  D2  C5  D4  D2  C9
0338-  C5  D3  BD  A0  A0  A0  A0  A0
0340-  A0  A0  A0  A0  D0  D2  C5  D3
0348-  C5  CE  D4  A0  D4  D2  C1  C3
0350-  CB  A0  A0  A0  A0  A9  30  38
0358-  ED  78  05  CD  A9  03  90  21
0360-  8D  A9  03  48  4A  4A  4A  4A
0368-  09  B0  C9  BA  90  02  69  06
0370-  8D  0D  04  68  29  0F  09  B0
0378-  C9  BA  90  02  69  06  8D  0E
0380-  04  CE  78  05  60  48  4A  48
0388-  4A  4A  4A  4A  09  B0  C9  BA
0390-  90  02  69  06  8D  24  04  68
0398-  29  0F  09  B0  C9  BA  90  02
03A0-  69  06  8D  25  04  68  4C  8E
03A8-  BE  0C
```

程式3－1的機械碼

這是一個 DOS 修改的例子。程式中利用到兩個 DOS 位址 $BDC9 及 $BE6D。前者本來是一個將$578內含值減一的指令（BDC9： DEC $578），程式3－1第8至13行將它改成BDC9： JSR CHECK（$352），而在 CHECK 的最後（52行）又補回原動作（即DEC $578），如此 DOS 進行到 $BDC9時便轉往 CHECK，而做完 CHECK 後，也完成了它應作的 DEC $578的任務了， DOS 的功能因此而不受影響。那

麼 CHECK 的功能是什麼呢？前面提到， DOS 有一個計數器（就是$578）用來計算磁碟機嘗試讀取的次數，而這個計數器是倒數計數的（也就是 BDC9 的原動作）最初的值為 $30 。程式 32 至 34 行用 $30 來減$578內含值，相減的差便是嘗試的次數。由 38 行至 54 行是用來將 16 進位值印在螢幕的最上方， 47 行以前印高半位元組，以後印低半位元組。由於 DOS 每讀寫一磁區均重新計數一次，如果每次印出計數值，結果將會顯示太快而看不清楚，程式 35 ～ 37 行便是用來比較每一磁區的嘗試次數，較小的值便不印出來，所以螢幕是清楚的計錄著最大嘗試值：

```
DATA RETRIES=09          PRESENT TRACK 03
```

至於 59 行的 RECAL 是用來印出目前所讀寫磁軌的號碼，軌道號碼（半軌）由 BE6D 進來時是存於暫存器 A，除 2 後便可印出來了。 79 行的 JMP $BE8E 是繼續 $BE6D 原來的流程。

本程式可以 CALL 768 或 300G 執行，由 HELLO 來 BRUN 也可以。本程式可與任何不使用第 3 頁的程式共容，更換磁片時也不用重新執行，一旦發現顯示值過大（超過 $10 ），那便是目前正使用的磁片有問題了。

3－3 DISK FULL?

對於資料處理者另外有一常見的 ERROR 叫做 DISK FULL，我們常會希望知道到底磁片還有多少空間可用？如果資料寫入途中發生 DISK FULL 豈不麻煩。雖然許多 COPY 程式均附帶有 FREE SPACE（檢查剩餘空間）的功能，但特別因此而另外執行程式甚或更換磁片則未免太過麻煩。此處我們介紹一個有趣的程式，每次執行 CATALOG 時，順便會在原來磁片號碼（VOL #）的位置上印出剩餘磁區數目，說來也是相當方便的。本程式相當簡短，總共才 39 個位元組，放在 DOS 空檔的 $BA69 區間裏是很適當的。

使用這個程式時，必須將 DOS 修改（ADC0： 20 69 BA）然後 BA69G [RETURN] [RESET] CATALOG 即可，如果在 HELLO 程式附加上：

```
0  REM *******************************
1  REM *    HELLO!!!  FREE  SPACE !    *
2  REM *******************************
3  POKE 44480,32: POKE 44481,105: POKE 4
     4482,186
4  PRINT  CHR$ (4)"BLOAD FREE_SPACE.OBJ"

5 HD$ = "FREE_SPACE:"
6  FOR I = 1 TO 11: POKE 46011 - I, ASC
     ( MID$ (HD$,I,1)) + 128: NEXT I
```

*請將程式 3－2 的機械碼 BSAVE FREE_SPACE.OBJ，A$BA69，L$27

則每次開機後，CATALOG 便有 FREE SPACE 的功能了。由於本程式以標準 DOS 爲準，對於某些特殊 DOS （LISA？），則要配合修改才行。

```
********************************
*   FREE SECTORS ON DISK    ...    *
********************************
*
             ORG  $BA69
*
FRCOUNT      EPZ  $0040
VOLNUM       EQU  $B3F2
LINPRT       EQU  $ED24          ；印出A及X的值
             LDA  #$00
             STA  FRCOUNT        ；歸零
             STA  FRCOUNT+1
             LDY  #$8C           ；由後面往前查
NEXT         CLC
             LDA  VOLNUM,Y
NEXTBIT      BEQ  NEXTBYTE       ；空格位元組
             ASL                 ；取一位元至C旗號
             BCC  NEXTBIT        ；若佔用（C＝1）
             PHA                 ；若爲空磁區
             INC  FRCOUNT        ；則加一
             BNE  CONTINUE
             INC  FRCOUNT+1
CONTINUE     PLA
             CLC
             BCC  NEXTBIT
NEXTBYTE     DEY
             BNE  NEXT
             LDX  FRCOUNT        ；取計數值
             LDA  FRCOUNT+1
```

```
        JSR LINPRT       ;印出來
        RTS
        END
```

程式 3－2　自動印出自由磁區的程式

```
*BA69.BA8F

BA69- A9 00 85 40 85 41 A0
BA70- 8C 18 B9 F2 B3 F0 0E 0A
BA78- 90 FB 46 E6 40 D0 02 E6
BA80- 41 68 18 90 F0 88 D0 E9
BA88- A6 40 A5 41 20 24 ED 60
```

程式 3－2 的機械碼

此類程式很多，但均大同小異，其方法主要是檢查 VTOC 位元映射表的內容，若位元值爲 1 則表示一個空白磁區，否則即爲已被佔用的磁區。 DOS 有一頁區（$B3BB～$B4BA）專門用來存放VTOC的內容（詳第 4 章），由第二章的說明來推算位元映射表恰好在 $B3F3～$B47E ，程式 3 － 2 便是用VOLNUM（$B3F2 ）及 LDY #$8C來查位元映射表。整個程式相當簡單，請參照程式註解即可明瞭，其中LINPRT（$ED24 ）是 BASIC 解譯器的副程式，用來將A及X的值以 10 進位方式印出。

3－4 DOS DOWN 掉怎麼辦？

有時候，我們會鍵入一個很長的 BASIC 程式，然後才發現沒有啓動 DOS ；有時候新程式偵錯執行一半而發生錯誤將DOS DOWN 掉。這些情形最麻煩的是一方面無法將程式存入磁片，而以PR＃6 來重叫 DOS 又會將 BASIC 程式清除，這種情況怎麼處理呢？

DOS 之所以會清除 BASIC 程式，一方面是由於其初值設定工作，而更重要是其載入過程中使用了＄800 以後的BASIC 程式使用區。如果將BASIC 程式在 DOS 未載入之前先移到其它安全區間，而當 DOS 載入完畢後再移回原位，程式便可保留了。DOS 載入過程中，未被使用到的安全區間是＄5000 至＄9000 ，只要程式長度不超過此範圍，便可使用。

可是，當 DOS 載入、程式移回原位時，你會發現程式不但無法執行，並且不能SAVE 到磁片！這是因爲程式之SAVE與RUN 均依靠 0 頁指位器來做指示（第一册），而 DOS 載入過程是將這些位址也清除的，所以程式便無法動作了。因此，除了程式本身外，0 頁位址也要保護才行。我們的方法是：

①記下 0 頁位址程式終點指位器（＄AF，＄B0）的內含值，並計算程式長度： 終點－＄800＋1＝長度 。

②移動程式：5000＜800.終點位置 M。

③保存BASIC 的 0 頁指位器：9060＜60.FFM （假定程式不超過 9060 ）。

④C600G 載入DOS。

⑤60 < 9060.90FFM，取回0頁指位器。

⑥800 < 5000.5000＋長度M，CTRL-C RETURN

這樣程式便可安全保存下來了。至於LISA系統，已於第0章提及，是比BASIC 更容易的。其實對於某些加上保護的BASIC 程式，也可以用這種方法加以拷貝。

參考文獻：

1. CALL-APPLE All about DOS 波前
2. APPLE Ⅱ微電腦徹底研究㈠，71，波前
3. APPLE Ⅱ BASIC REFERENCE MANUAL
4. LISA活用法，72，波前

本章提要

從這一章開始，我們要進入 DOS 內部，探究其構造。DOS 的構造，除了程式本身外，還有各種向量、工作空間、暫存區……。因此：

①DOS 程式的三大部份：DCI，FMS，RWTS

②系統暫存區與檔案暫存區

③第 3 頁向量

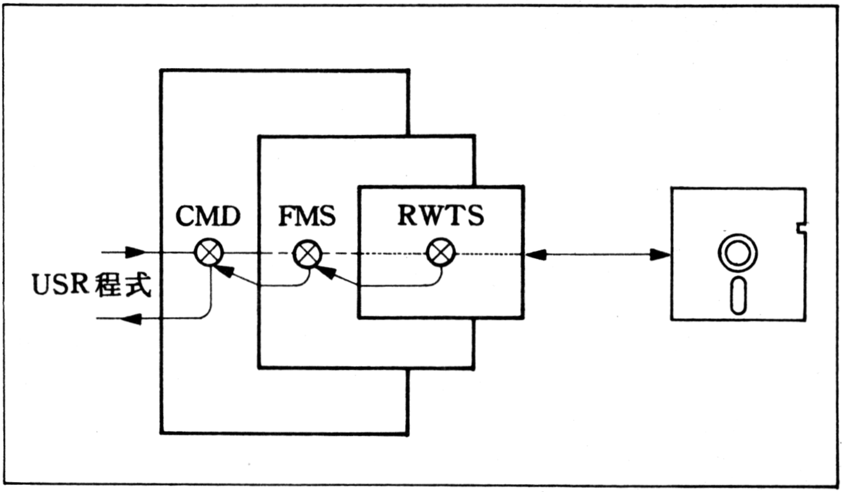
④DOS 的記憶結構

⑤FMS 的使用法

⑥RWTS 的 BASIC、機械語與各種應用

均是本章的主題！

4 解剖DOS



知道了檔案在磁片上的安排之後，接下來的工作，便是如何將磁片上的檔案系統來與APPLE 主機聯繫起來。這個聯繫及管理磁碟系統的程式就是DOS……

4－0 DOS的組成

基本上DOS 程式（＄9D00～＄BFFF）是由命令解譯器（DOS Command Interpreter ，簡稱DCI）、檔案管理系統（File Management System ，簡稱 FMS ）及磁軌磁區讀寫程式（Read Write Track Sector，簡稱RWTS ）所構成的。

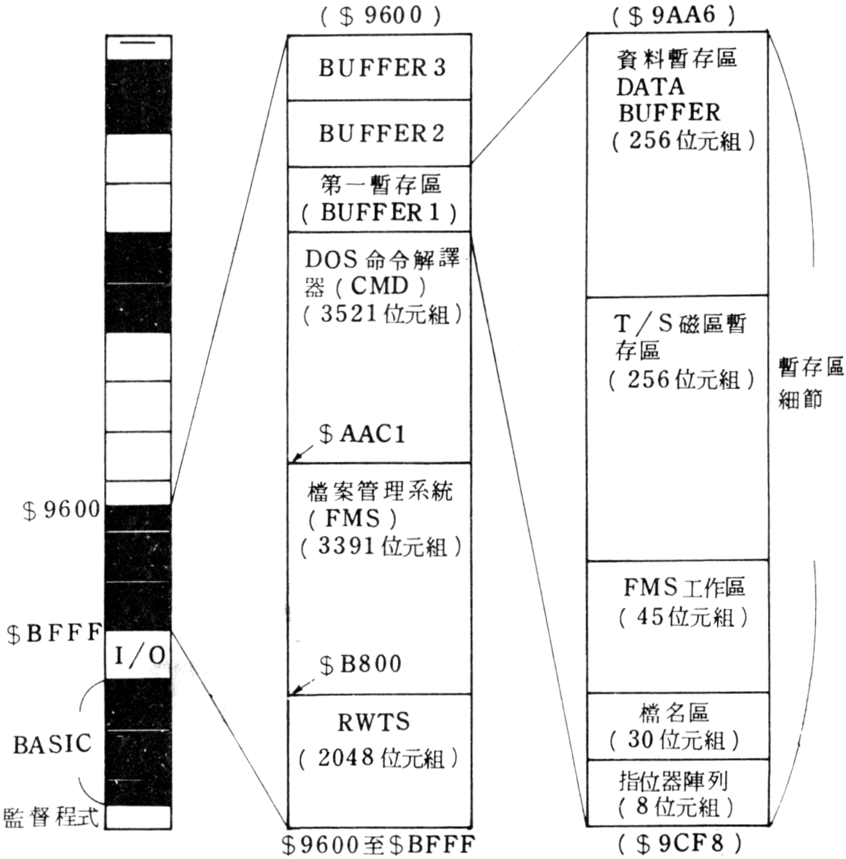


圖 4－0 APPLE DOS的記憶結構

除了程式本身之外，當然還有零頁位址以及一些專屬 DOS 的工作空間，這些包括了第三頁的 DOS 向量及圖 4－0當中的檔案暫存區(通常是 $9600 至 $9CF8)。

命令解譯器(DCI)是用來攔截 APPLE 命令，以免DOS命令進入BASIC 解譯器而誤解成SYNTAX ERROR(因爲BASIC 解譯器根本不認識CATALOG這種DOS 命令)，命令

欄截之後便送往命令分析程式。命令分析無誤後，DCI 首先將命令訊息設定於檔案管理系統（FMS）的使用表（parameter list）用以傳達命令給 FMS 。FMS 便會依使用表所指示的內容來處理檔案，FMS 的處理過程（OPEN、CLOSE、READ、……等）大致上就如第一章所述一般。一旦 FMS 需要 I／O 處理的時候（共 SEEK、READ、WRITE 及 FORMAT 四種），便將 I／O 命令訊息設定於RWTS 的使用表（I／O CONTROL BLOCK 簡稱 IOB），然後呼叫 RWTS 來做 I／O 工作。這便是 DOS 的工作概況，其相互間關係請參照本章篇頭圖示。大致說來由 RWTS→FMS→DCI 其邏輯層次漸高，DCI 負責使用者介面、WRTS 負責實體 I／O，而 FMS 則介於其間。

4－1 VTOC與目錄暫存區：哪些東西需要搬進APPLE？

由於磁片是一被動元件，任何資訊之處理均須由主機來決定，因此，在主機內部，必須保持著磁片最新與最重要的關鍵情報。什麼是關鍵情報呢？由第一、二章，我們知道磁片的重要情報包括有：資料磁區、T／S 表、FDB 、目錄表、VTOC 等。

其中，VTOC 及目錄表是有關於整體磁片的總情報，因此，由 DOS 來統一管理。對於 VTOC，DOS 是以位於＄B3BB～＄B4BA 共 256 位元組的空間來存放對應於磁片（＄11，＄00）磁區的資料，此區間隨時反應著VTOC 的每一位元組細節，上一章的自由空間（FREE SECTOR）程式便是取這個暫存區的資料來計算的。至於目錄表，由於是由 15 個磁區串聯起來，若全部拷貝（存）到主記憶體來是相當浪費空間的，並且實用上的磁

片，檔案甚少超過 2 個目錄磁區，因此也用256 位元組的空間來存放目前所使用檔案所在的目錄磁區，其位置是由＄B4BB ～＄B5BA 。另外，DOS 使用（AAC3，AAC4 ）及（AAC5，AAC6 ）這兩組指位器來指向 VTOC 及目錄磁區所在的位置。

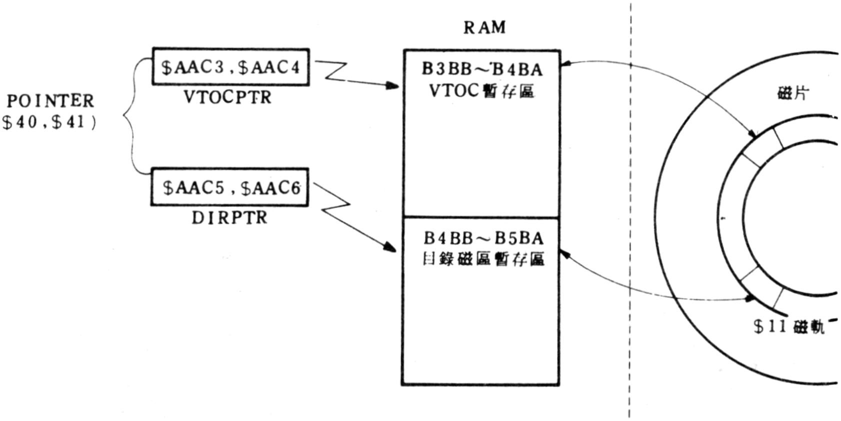


圖 4－1 VTOC 及目錄磁區存放於 RAM的情形

4－2 檔案暫存區：BUFFER

上述的 VTOC 及目錄表是屬於磁片整體性的資料*，但是其他如 FDB、T／S 表、資料磁區等均是分屬每個檔案而各不相同的。如果磁片有 10 個檔案，則就要有 10 個不同的區間來存放這些各別資料，如此勢必大量佔用了記憶空間，因此 APPLE 在正常情況下只分配了 3 個檔案存放資料的位置，稱為檔案暫存區（

---

* 即系統暫存區

BUFFER ），其它檔案則不存到主記憶體內。

檔案暫存區的內容是什麼呢？由於考慮了佔用空間的問題，因此暫存區內也只用了兩個256位元組的空間分別來存放一個T／S磁區及一個資料磁區，由於一個T／S磁區包含了122個資料磁區指位器，因此較少被交換（SWAP，交換另一個T／S磁區），但是一個資料磁區顯然是不足使用的，因此常被交換而影響了系統的效率，這是DOS 的瓶頸之一。

至於 FDB ，其中的檔案名稱被存於30 位元組的檔案名稱暫存區，其他的 FDB 資料，則分散存於一個 45 位元組的工作區（Work Area ） 內，因此整個架構是這樣的：

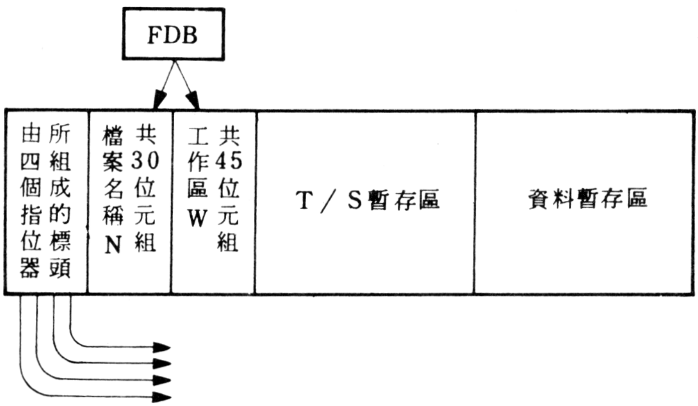


圖 4 － 2　一個檔案暫存區的構造

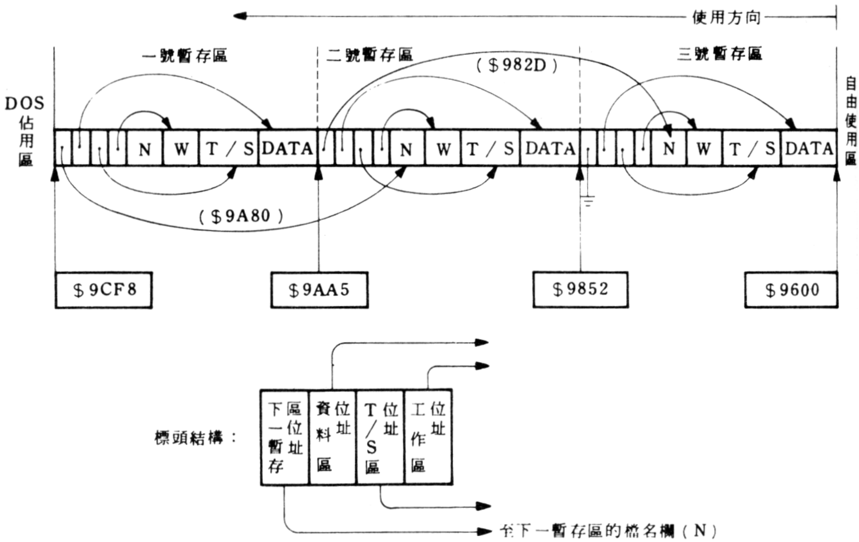


圖 4 － 3　由 3 個暫存區所組成的鏈(請注意位址高低關係)

到底圖 4 － 3 的細節是怎樣安排的呢？首先我們寫一個程式來試驗一下吧！

```
5 D$ =  CHR$ (4)
10   PRINT D$"OPEN TEST 4.0"
20   PRINT D$"WRITE TEST 4.0,B0"
30   PRINT "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
40   PRINT "1234567890"
```

程式 4 － 0

首先由第 10 行 OPEN 一個叫 TEST 4.0 的檔案，然後寫入 26 個英文及 10 個數字。接著我們進入監督程式來看看暫存區的情形：

```
*982D.9852
                                  ?EST 4.0
982D-  00 C5 D3
9830-  D4 A0 B4 AE B0 A0 A0 A0
9838-  A0 A0 A0 A0 A0 A0 A0 A0
9840-  A0 A0 A0 A0 A0 A0 A0 A0
9848-  A0 A0 A0 00 98 00 97 00
9850-  96 00 00
```

圖 4 － 4　第 3 檔案暫存區的檔名區

因爲暫存區有 3 個，首先要確知 TEST 4.0 到底是存入哪一個？結果在第 3 暫存區的檔名區（＄982D ～＄9852 ）內找到了 TEST 4.0這個名字。因此APPLE DOS 是第 3 暫存區最先被使用，接著才是第 2 第 1 暫存區。更値得注意的是檔名區也是依圖 4 － 3 的使用方向（低往高）來塡入各個檔名字元的，這與標頭指位器的方向是完全相反的，光看圖 4 － 3 很易誤解。

再進一步追查，我們發現資料眞的寫在第 3 資料暫存區（＄9600 、96FF ）， 並且也是由低位址往高位址塡入：

```
*9600.96FF

9600-  C1 C2 C3 C4 C5 C6 C7 C8
9608-  C9 CA CB CC CD CE CF D0
9610-  D1 D2 D3 D4 D5 D6 D7 D8
9618-  D9 DA 8D B1 B2 B3 B4 B5
9620-  B6 B7 B8 B9 B0 8D 00 00
9628-  00 00 00 00 00 00 00 00
9630-  00 00 00 00 00 00 00 00
9638-  00 00 00 00 00 00 00 00
9640-  00 00 00 00 00 00 00 00
```

圖 4 － 5　第 3 檔案暫存區的資料區

其中 8D 是 PRINT 指令結束時自動送出的 RETURN 碼，通常我們以這個碼來做爲檔案裡錄（ Record ）與錄之間的間格指示字元。

再來是由＄9700 至＄97FF 的T／S表暫存區，由於 TEST 4.0 的內容太短，只使用一個磁區，因此T／S表就只有（＄10 ，＄0E ）這個指位器了。並且依照第二章的說法，這個指位器是位於（＄970C ，＄970D ）：

```
*9700.977F

9700- 00 00 00 00 00 00 00 00
9708- 00 00 00 00 10 0E 00 00
9710- 00 00 00 00 00 00 00 00
9718- 00 00 00 00 00 00 00 00
9720- 00 00 00 00 00 00 00 00
9728- 00 00 00 00 00 00 00 00
9730- 00 00 00 00 00 00 00 00
9738- 00 00 00 00 00 00 00 00
9740- 00 00 00 00 00 00 00 00
```

＊T／S表會因爲磁片使用歷史而有所不同

圖 4－6　第 3 暫存區的T／S區

至此，或許你已發現，爲何檔名區開頭的元組內含值是 0 呢？檔案名稱的第一個字母T竟然不見了！請 DELETE 掉 TEST 4.0 ，然後執行下列程式：

```
5 D$ =  CHR$ (4)
10  PRINT D$"OPEN TEST 4.0"
20  PRINT D$"WRITE TEST 4.0,B0"
30  PRINT "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
40  PRINT "1234567890"
50 X =  PEEK ( - 26579): REM  $982D
55  FOR I = 0 TO 5000: NEXT I
60  PRINT D$"CLOSE TEST 4.0"
70  PRINT  PEEK ( - 26579)
80  PRINT X

]RUN
0
212
```

程式 4 － 1

這個程式的 50 及 70 行分別將檔案 CLOSE 前後的檔名區最前頭位元組的值印出來，我們可以看到在未 CLOSE 前，其值為 212，而 CLOSE 後就變成 0 了。雖然程式 4 － 0 並沒有做 CLOSE 動作，但 DOS 在程式結束時會 CLOSE 檔名區，因此其值就變成 0 了（圖 4 － 4），將程式 4 － 1 DEL 55，80 之後再執行一次：

```
]RUN
PRINT X
212
]PRINT PEEK(-26579)
0
```

程式 4 － 2

因爲X值是由 50 行在程式未結束時記錄下來的，所以不爲0，而PRINT PEEK（－26579）取的是程式結束後的值，因而是 0，我們的說法便清楚的印證了。原來每次檔案OPEN後，其名字就填入檔名區，而一旦程式結束或檔案關閉時，DOS 便將檔名區的起始位元組設爲0，並由暫存區檢查此位址便可以知道在此暫存區上的檔案是否已CLOSE 。程式設計者直接使用暫存區時，要記得先檢查檔名暫存區第一位元組是否爲0，若是，則填入檔案名稱，以佔用此區的使用權，使後來的檔案不得重覆使用；而當結束使用暫存區時，要將檔名暫存區的第一位元組設爲 0，以使其它檔案可以使用此區。

平常暫存區有3組（MAXFILES 3），互相間以指位器來連結。每一組暫存區均有一個由4個指位器構成的標頭（Header），分別指向資料區、T／S區、工作區（W）以及下一檔名區（N）的開頭位址（開頭在低位址那端）。若下一暫存區已被使用（或最後一個），則下一檔名區指位器爲0（nil）。

DOS指令使用到暫存區的情形

使用到暫存區	只有執行過程使用到		不使用
OPEN	INIT	DELETE	PR#
APPEND	LOAD	RENAME	IN#
EXEC	SAVE	VERIFY	MON
READ	RUN	CATALOG	NOMON
WRITE	BLOAD	CHAIN	FP
POSITION	BSAVE		INT
	LOCK		MAXFILE
	UNLOCK		CLOSE

最後剩下 45 位元組的工作區（W），TEST 4.0的工作區位於：

```
*9801.982C

9801- 0F 0D 0F 02 10 0E 00
9808- 00 7A 00 00 00 7A 00 00
9810- 00 00 01 00 00 26 00 01
9818- 00 26 00 00 00 02 00 0E
9820- 00 3F FF 00 00 00 60 01
9828- FF 11 00 00 00
```

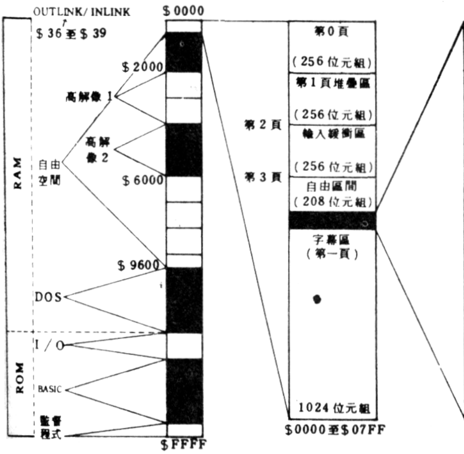
圖 4 － 7　第 3 工作區

其內容所代表的意義請參照圖 4 － 8 。比照之後，我們可以知道 TEST 4.0 這個檔案的 T／S 表由（ 0D，0F ）開始，目前的 T／S 磁區也是（ 0D，0F ），VTOC的 B－map 已更新（02），在暫存區內的資料磁區爲（ 10，0E ）……。因爲程式 4 － 1 磁碟機的最後動作是 CLOSE ，因而磁頭的最後動作是至目錄表、VTOC 等位置更新資料，所以目前（最後）磁軌是＄11，而目前磁軌的位元映射是00 00（＄9823、4 ）。 FMS 是以此區做爲其工作資料的暫存區，一切動作皆記錄於此，所以才稱爲工作區。由於程式設計時只要控制使用表（ parameter list ，後述）便能操縱 FMS ，因此工作區除了可由檢視其內含值而得知檔案狀況外，對使用者而言，較不常使用。

	位元組	用　　途
絕對位置座標	00－01	本檔案T／S表的開始位置（T，S）
	02－03	目前載入的T／S磁區位置
	04－04	旗號：80＝T／S區內容已更新，40＝資料區已更新 20＝VTOC區內容已更新，02＝上次的動作是'Write'
	05－06	目前載入的資料磁區位置
目錄索引	07－07	用來由目錄表中找出'檔案所在目錄磁區'的索引
	08－08	用來由目錄磁區中找出'檔案所在FDB'的索引
T／S表內資料	09－0A	一個T／S磁區所含的磁區數（指位器個數），通常為122（7A）個
	0B－0C	目前載入的T／S磁區第5、6位元組值，即T／S磁區的磁區抵補值。
	0D－0E	目前T／S磁區所含最後磁區的距離抵補（與檔案第一磁區的距離）
	0F－10	與前一次所讀磁區的距離（單位：磁區）
檔案內位置	11－12	磁區長度（256位元組）
	13－14	目前讀寫位置與檔案開頭的距離（單位：磁區）
	15－15	目前讀寫位置與該磁區開頭的距離（單位：位元組）
	16－16	未　使　用
檔案內位置（錄為單位）	17－18	OPEN時所設定的'錄'長度（單位：位元組）
	19－1A	目前所在（讀寫）'錄'的號碼
	1B－1C	目前所讀寫錄內的位置（單位：位元組）
	1D－1E	檔案長度（單位：磁區）
磁區配置情形	1F－1F	下一個要使用的磁區（寫）
	20－20	目前正使用的磁軌
	21－24	目前正使用磁軌的磁區位元映射表
檔案及系統狀態	25－25	檔案類型及上鎖旗號
	26－26	插座號碼乘16
	27－27	磁碟機號碼
	28－28	磁片號碼
	29－29	上次使用的磁軌號碼
	2A－2C	未　使　用

圖4－8　工作區各位元組的用途

4－3 第三頁向量

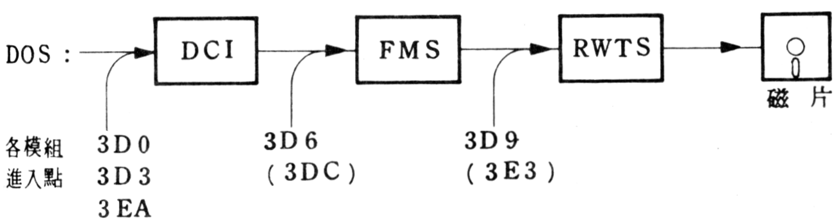


10進位	第三頁向量	16進位	正常內含值	
	DOS 向量			
976	DOS及BASIC熱啓動向量	3D0	JMP	$9DBF
979	DOS及BASIC冷啓動向量	3D3	JMP	$9D84
982	FMS向量	3D6	JMP	$AAFD
985	RWTS向量	3D9	JMP	$B7B5
988	FMS使用表位址向量	3DC	LDA LDY RTS	$9D0F $9D0E
995	RWTS使用表（IOB）位址向量	3E3	LDA LDY RTS	$AAC2 $AAC1
1002	DOS欄截重置向量	3EA	JMP	$A851
	監督程式向量			
1007	BRK處理向量	3EF	JMP	$FA59
1010	RESET向量	3F2	$9DBF	
1012	冷開機檢知位元組	3F4	38	
1013	&向量	3F5	JMP	$FF58
1016	CTRL-Y向量	3F8	JMP	$FF65
1019	NMI向量	3FB	JMP	$FF65
1022	IRQ向量	3FE	$FF65	
	共 48 位元組			

$03D0 至 $03FF

圖 4－9　第 3 頁向量的記憶結構

由於 DOS 的構造複雜，直接叫用或修改其中副程式，往往會超乎預想的發生一些麻煩。不過APPLE 的 DOS 還算是模組化的，其 3 層模組（ DCI → FMS → RWTS ）均很完整，如果由這些模組程式的正常進入點來呼叫使用，則便能安全的控制DOS而無意外了。



可是這些模組的進入點在哪裡呢？湊巧的是，由於 APPLE 早期是分成 16、32、48K 等不同的記憶量，而 DOS 又是存在於記憶體的高位址區，如此一來，各模組進入點就會因記憶量的多寡而發生差異了。因此 DOS 設計時，便將各模組進入點集中於第 3 頁區，而稱爲第 3 頁的 DOS 向量。將進入點至於第 3 頁的好處是，當使用不同記憶量時，進入點雖然改變了，但向量位置却不改變（只向量值自動跟著改變）：

	向量值			進入點	
向量位置 3D0G	$ 1DBF ……>	16K ……>		$ 1DBF	DOS
	$ 5DBF ……>	32K ……>		$ 5DBF	
	$ 9DBF ……>	48K ……>		$ 9DBF	

對於現今 48K以上的使用者而言，第 3 頁向量仍然有其容易記憶的好處（只兩位數要記因均爲 3 開頭）。 DOS 的第 3 頁向量詳述如下：

3D0 ：這時 DOS 熱啓動位址。3D0G 後，控制權便交給 DOS ，但記憶體內的BASIC 程式仍舊保存不消失。這個位址的用處並不在立即指令狀態，因爲 RESET 向量依然可以將 DOS 啓動。其用途是在程式執行當中，以CALL 976 很方便的可以回到 DOS 。

3D3 ：與 3D0 相同，但記憶體內的BASIC 程式同時也被消除掉。

3D6 ：進入檔案處理系統 FMS 。在進入 3D6 之前，要先設定好檔案處理器的使用表，然後JSR 3DC將使用

表傳給 FMS ，最後才可進入 3D6 。

[3D9]：進入磁軌磁區讀寫程式 RWTS 。在進入 3D9 之前，要先設定好讀寫程式使用表（ IOB ），然後 JSR 3E3 將使用表傳給 RWTS ，最後才可進入 3D9 。

[3DC]：將 FMS 使用表傳給 FMS 。由 3DC 返回後，使用表的高位址會存入A暫存器，低位址會存入Y暫存器。

[3E3]：將 RWTS 使用表（ IOB ）傳給 RWTS 。由 3E3 返回後， IOB 的高位址會存入A ，低位址會存入Y 。

```
   :                 :
   :                 :
設定 IOB           設定 IOB          (A , Y)→ [ IOB ]
JSR  3E3           A←/ IOB
JSR  3D9           Y←# IOB
BCS  ERR           JSR  3D9
   :               BCS  ERR
   :                 :
(使用系統原          :
來的 IOB )         (自設 IOB )

* IOB及第3頁向量使用法見4－6節
```

程式 4－3　RWTS使用法（ FMS類似）

4－4 DOS的記憶結構要點

介紹了 DOS 所使用的各暫存區、第 3 頁向量之後，此處將其相互之間的關係做一整理，希望建立一明析的整體概念，以利於往後各章的細部討論。

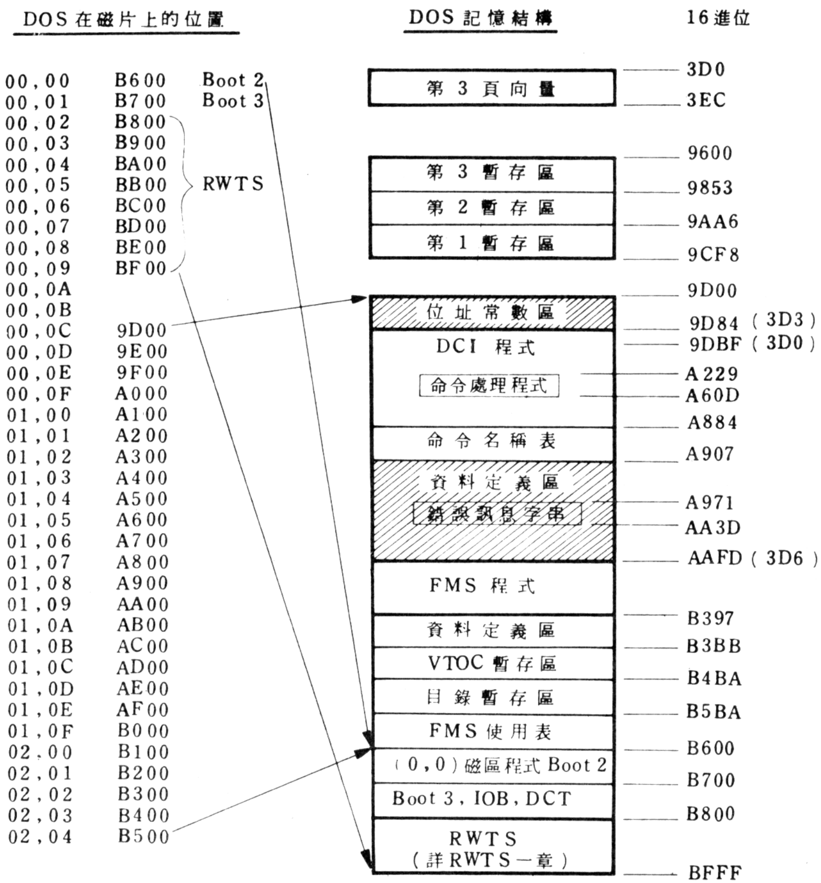


圖 4－10 DOS的記憶結構

由上圖我們可以看出，DOS 系統最前頭的是第3頁向量，由9600 至9CF8是3個檔案暫存區，DOS 程式則由9D00開始。DOS 本身又分成3段， DCI 由9D00至AAC8，其中包含了各資料定義區(存放各常數、變數、指位器等)、命令名稱表、錯誤訊息字串等。 FMS 程式由AAC1(C1～C8 爲DCI變數)至B5FA，其中包含了磁片的VTOC及目錄暫存區、FMS

使用表、資料定義區等，B600 至 B7FF 爲第 2 、 3 階段的DOS 載入程式。而 RWTS 則由 B800 至 BFFF。至於圖左是 DOS 在磁片上位置的對照表，對於修改（或保護）DOS 很有幫助。

檔案暫存區的利用

由於檔案暫存區具有 595 個位元組空間，並且 DOS 會設定 HIMEM 使得 BASIC 程式不會侵入到此區間，因此若將我們的機械語言程式藏入此處是很理想的，本書的 RAM＿DISK 程式便是這種做法。

使用暫存區的方法是這樣的：

```
;
DOS       EQU $9D00          ; 第 1 號檔名區位址
MAXFILE   EQU $AA57          ; MAXFILE 現值
BUILDBUF  EQU $A7D4          ; 設定 BUF 連結副程式
TARGET    EQU $9AA6
*          .
*          .
*          .
SETBUF    LDA #2             ; 佔用一個 BUF
          STA MAXFILE
          LDA #TARGET-38     ; 低位址
          STA DOS
          LDA /TARGET-38     ; 高位址
          STA DOS+1
          JSR BUILDBUF       ; 將第 1 號 BUF 由
*          .                 ; BUF 連結中去除
*          .                 ; 並設定 HIMEM
*          .
          END
```

程式 4 － 4　暫存區的使用法

程式中，TARGET 是第一暫存區的最高位址，由於 DOS = $9D00 存的是第一檔名區的起始位址，與最高位址差 38 個位元組，必須將檔名區位址存入 9D00 BUILDBUF 才會正確工作。

其他可使用的空間

除了較長的程式外，其實是不必要耗費檔案暫存區空間來存放程式的。在 DOS 當中，尚有許多空間，我們可以用來填入程式，這對於一些 DOS 的小修改 5－3 節，是很方便使用的。

空區位址	長度
$ B6B3 – B6FC	73 位元組
$ BA69 – BA95	45 位元組
$ BC56 – BCFF	169 位元組

上述的空區雖然各自獨立，其實也可以用 JMP 指令連接起來。這些區間與第 3 頁差異之處是在：前者可以由 INIT 將程式隨同 DOS 存入到新的磁片上（或用COPY 式寫入到磁片的 0 － 2 號磁軌），因此每次 Boot ，程式也一併載入到主機來。下一節的程式也有這種 DOS 修改的功能。

4－5 FMS的使用方法

使用 FMS 要先設定一個 FMS 的使用表（ parameter list ），所有我們要令 FMS 做的動作均要設定於使用表上，然後才可呼叫 FMS 。由 DOS 的第 3 頁向量我們知道 JSR 3DC 可以將DOS 的 FMS 使用表位址存入A（高）及Y（低）暫存

器內，對於 48K 的正常 DOS ，其動作是：

```
03DC-    AD 0F 9D    LDA   $9D0F
03DF-    AC 0E 9D    LDY   $9D0E
03E2-    60          RTS
```

所以使用表位址其實是存於指位器（ 9D0E ， 9D0F ）：

```
*9D0E.9D0F

9D0E- BB B5
```

由此可看出，使用表的起始位址是$B5BB，共佔有 18 個位元組。程式設計者也可以在其他位址另造使用表，通常在造好使用表之後，要將起始位址存於 0 頁指位器，以方便用索引法來存取使用表內容。如果是使用 DOS 原來的 FMS 使用表，將起始位址存入 0 頁指位器時，可以直接用 B5BB。但爲了顧及系統記憶體大小不同的場合，通常是使用 3DC 向量，再存入 0 頁位址：

```
JSR   $3DC       ;取$B5BB
STY   PTR        ;低位址
STA   PTR+1      ;高位址
```

FMS使用表

與傳統的系統呼叫（ system call ）比較起來，FMS 使用

表要設定的參數是太多了一點，因此先逐一介紹如下表：

位元組	用　　途
00－00	呼叫（指令）碼，參照圖 4 － 13 第2行。
01－01	次呼叫碼，只在READ及WRITE 使用。00＝NOP；01＝在目前位置上R／W一個位元組；02＝由目前位置開始R／W一串位元組；03＝移動至目的位置上，然後R／W一個位元組；04＝移動至目的位置上，然後R／W一串位元組。
02－09	存放R、B、V、D、S等參數值、檔名位址、欲讀寫資料的起始、終點位址、檔案型式等，參照圖 4 － 12 。
0A－0A	返回碼，由這個碼我們可以知道 FMS 執行後的情形。若返回碼≠0，則C旗號同時也會被設爲1（錯語碼）： 00＝執行無誤　01＝LANGUAGE NOT AVAILABLE 02＝不正確的呼叫碼（參數）　03＝不正確的次呼叫碼（>4） 04＝WRITE PROTECTED　05＝END OF DATA 06＝FILE NOT FOUND　07＝VOLUME MISMATCH 08＝DISK I/O ERROR　09＝DISK FULL 0A＝FILE LOCKED 這是DOS錯誤訊息中與FMS有關者
0B－0B	未　使　用
0C－0D	工作區起始位址
0E－0F	T／S暫存區的起始位址
10－11	資料暫存區的起始位址

圖 4 － 11　FMS使用表的說明（Ⅰ）

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
OPEN	01		資料錄長或 0000		V	D	S	檔案類型	檔案名稱之位址		返回碼（輸出）		檔案處理器工作暫存區之位址		T／S表暫存區之位址			
CLOSE	02																資料段落暫存區之位址	
READ	03	次呼叫碼	資料錄號數		位元組移動量		資料區長度		一位元組或位元組串的位址									
WRITE	04																	
DELETE	05				V	D	S		檔案名稱之位址									
CATALOG	06					D	S											
LOCK	07				V	D	S		檔案名稱之位址						T／S表位址			
UNLOCK	08				V	D	S											
RENAME	09		新名稱位址		V	D	S											
POSITION	0A		資料錄號數		位元組移動量													
INIT	0B	DOS頁數			V	D	S											
VERIFY	0C				V	D	S		檔案名稱之位址						T/S表暫存區位址		資料暫存區之位址	

圖 4 － 12　FMS使用表的說明（Ⅱ）

FMS使用例

```
0   REM  ******************************
1   REM  *  PATCHING  DOS  ..........   *
2   REM  ******************************
10  FOR I = 32768 TO 32822: READ A: POKE I
    ,A: NEXT
20  HOME : TEXT : INPUT "ENTER THE NAME OF
    THE GREETING PROGRAM";A$
30  IF  NOT  LEN (A$) THEN 20
40  T =  ASC ( LEFT$ (A$,1)): IF T < 64 OR
    T > 90 THEN  PRINT "ILLEGAL FILENAME!
    ": GOTO 20
50  POKE 32792, LEN (A$) - 1
```

```
55  REM *** STORE THE FILENAME ***
60  FOR I = 32823 TO 32823 +  LEN (A$) - 1
     : POKE I, ASC ( MID$ (A$,I - 32822,1)
     ) + 128: NEXT I
105  REM ***STORE DOS ONTO THE DISK***
110  CALL 32768
1000  DATA   186,142,155,179,162,15,189,37
     ,128,157,187,181,202,16,247,169, 0,14
     1,95,170,32,149,160,160,4,185,55,128,
     153,117,170,136,16,247,76,255,174,11,
     157,1,0,254,1,06,0,117,170,0,0,0,152,
     0,151,0,150
```

程式 4 － 5　利用 FMS 來改寫磁片上的DOS程式

```
 2 **********************************
 3 * Using the FMS to patch DOS ... *
 4 **********************************
 5 *
 6 CMDINDX   EQU $AA5F
 7 FNAME     EQU $AA75
 8 WRITEDOS  EQU $AEFF
 9 STKSAVE   EQU $B39B
10 FMSPL     EQU $B5BB
11 BLKNAME   EQU $A095
12 *
13           ORG $8000
14           OBJ $6000          ; 避開LISA使用區
15 START     TSX
16           STX STKSAVE
```

```
17          LDX  #$0F
18 NEXT     LDA  DATA,X
19          STA  FMSPL,X            設定FMS使用表
20          DEX
21          BPL  NEXT
22          LDA  #$00
23          STA  CMDINDX            ;設命令碼爲INIT
24          JSR  BLKNAME            ;清除檔名區
25          LDY  #$04
26 NEXT1    LDA  NAME,Y
27          STA  FNAME,Y            填入暫存區的檔名區內
28          DEY
29          BPL  NEXT1
30          JMP  WRITEDOS
31 DATA     HEX  0B9D0100FE01060075AA00000098
                 00970096
32 NAME     DFS  30                 Greeting 檔名
33          END
```

程式4－6　修改DOS的機械語言程式（程式4－5的第1000行）

這是一個利用 FMS 將 DOS 程式寫入磁片 0～2 號磁軌的程式，它與 INIT 相似，但不做磁片格式化（ Format ）的工作，因而不會損壞磁片上的既存資料。這個程式對於將修改後的 DOS 存入磁片的場合很有用， DOS 的修改將於本章後述。

程式4－5第1000行的組合語言程式列於程式4－6，由這個程式我們可以看出 FMS 使用表的用法。程式中 31 行是要存入 FMS 使用表的資料，這些資料由 17 至 21 行來存入使用表。一一檢視這些數值可以看出＇0B ＇是INIT的呼叫碼、＇9D ＇是 DOS 開始位址也是我們要寫入磁片的起始位址、＇FE ＇是磁片號碼、＇AA75＇是歡迎（ Greeting ）程式的

檔名位址、……，如果能確知使用表內均爲合法值，則只要適當的改變所需參數便可以正確的運用使用表，而不必將 18 個位元組全部重寫。

本程式由於不做 Format 工作，所以由 30 行的 JMP WRITE DOS 來執行 INIT 的'部份工作'，而不直接呼叫 FMS。雖然不直接叫用 FMS，但是 WRITE DOS 也屬於 FMS 的一部份，也會用到使用表資料，所以與 FMS 一樣要先設定好使用表才能正確工作。

與下一節要介紹的 RWTS 比較起來，FMS 是很接近高階的邏輯層次，與 DCI 相去不遠，因爲在這個階層大部份檔案處理工作已由 FMS 完成，而 DCI 所做的只不過是命令的分析、解譯而已。由於 FMS 與 DCI 太過接近，使得直接叫用 FMS 並無多大用處，因此除非爲了修改 FMS 的功能（如上例），否則直接叫用 DOS 指令（由 DCI 處理）反而更方便簡潔。所以通常 DOS 使用的情形是：①在 BASIC 程式使用 DOS 命令來處理檔案。②在機械語言程式當中使用 RWTS 直接做 I／O 來快速讀寫磁片資料。FMS 通常是較少使用的。

4－6 RWTS的使用方法

呼叫 RWTS 時，也和 FMS 一樣要先設定一個 RWTS 使用表，這個使用表有一個常用的簡稱叫做 IOB（I／O control Block）。由於 RWTS 是眞正做 I／O 的程式，因而除了 IOB 外，還要設定一個設備使用表 DCT（Device Characteristics Table），這兩個表與 FMS 使用表一樣平時已由 DOS 設定完

成，程式設計者只要修改需要的位元組即可。

IOB

位元組	用　　途
00－00	表格型式，規定爲01
01－01	插座號碼乘以16
02－02	磁碟機號碼
03－03	磁片號碼，其值若爲00，則與任何號碼通用。
04－04	磁軌號碼（＄00－＄22）
05－05	磁區號碼（＄00－＄0F）
06－07	DCT的起始位址
08－09	資料暫存區位址
0A－0A	未　使　用
0B－0B	未　使　用
0C－0C	呼叫碼，00＝SEEK，01＝READ，02＝WRITE，04＝FORMAT
0D－0D	返回碼，若返回碼≠0，C旗號會被設爲1。 0＝無錯誤，10＝WRITE PROTECTED， 20＝VOLUMN＃MISMATCH， 40＝DRIVER ERR，80＝READ ERR （所有碼爲16進位）
0E－0E	上次的磁片號碼
0F－0F	上次的插座號碼乘以16
10－10	上次的磁碟機號碼

DCT

位元組	用　　　　　　　　途
00 − 00	設備型式，規定爲 00
01 − 01	磁軌間距與步級馬達相位的對應值規定爲 01
02 − 03	等候馬達穩定的時間，規定爲 EFD8

呼叫 RWTS 前，除了設定 IOB、DCT 外，還要將 IOB 的起始位址存於 A 暫存器（高）及 Y 暫存器（低）。如果你是使用 DOS 原來的 IOB 位址，則可以與 FMS 一樣的使用第 3 頁向量 JSR 3E3 來將位址取入 Y 及 A。當然 Y 與 A 的值也要存入 0 頁指位器以方便用索引法來存取 IOB 的內容。由 3E3 向量我們可以找到 IOB 的正常位置是 $B7E8（方法見 FMS）。

RWTS使用例(I)：更改VTOC

```
100  REM ******************************
101  REM **     DOS__KILLER  .....     **
102  REM ******************************
110  TEXT : HOME
120  PRINT "DOS-KILLER": PRINT "__________
     _": PRINT : PRINT "FREES 32 SECTORS
     FOR AN ADDITIONAL 6%": PRINT "OF STO
     RANGE,MAKING THE DISK UMBOOTABLE": PRINT
     "ONCE A PROGRAM HAS OVERWRITTEN DOS.
     "
140  PRINT : PRINT "INSERT DISK AND PRESS
      RETURN:";: GET A$: IF A$ < > CHR$
     (13) THEN 260
160  POKE 47084,17: POKE 47085,0: REM  SE
     LSCT TRACK 17, SECTOR 0
```

```
170  POKE 47083,0: POKE 47091,0
180 LOC = 10000: POKE 47088,LOC -  INT (L
     OC / 256) * 256: POKE 47089, INT (LO
     C / 256): REM  STORE DATA AT LOCATIO
     N 10000
190  POKE 768,32: POKE 769,227: POKE 770,
     3: POKE 771,76: POKE 772,217: POKE 7
     73,3: REM   ROUTINE FOR JUMP TO RWTS

210  POKE 47092,1: CALL 768: REM  READ SE
     CTOR
220  POKE LOC + 60,255: POKE LOC + 61,255
     : POKE LOC + 64,255: POKE LOC + 65,2
     55: REM  REWRITE 4 BYTES
230  POKE 47092,2: CALL 768: REM  WRITE S
     ECTOR
240  VTAB 3: PRINT : CALL  - 958
250  PRINT : PRINT "DONE.": PRINT : PRINT
     "UPDATE ANOTHER DISK?(Y/N):";: GET A
     $: IF A$ = "Y" THEN  RUN
260  VTAB 20: END
```

程式 4 － 7　增加磁片空間的程式

這是一個很有用的程式，它可以將 DOS 佔用的磁片空間讓出來給一般檔案使用，雖然 DOS 因為會被檔案資料蓋過而破壞，但是對於專用來存放資料（如Visicalc)的磁片而言， DOS 是不必要的，因此，請執行第 3 章的FREE SPACE，然後 LOAD DOS－KILLER，換一張空白磁片，INIT DOS－KILLER ，結果：

```
]CATALOG

DISK VOLUME 254         492 FREE SECTORS

 A 004 DOS-KILLER
```

RUN DOS－KILLER後：

```
]CATALOG

DISK VOLUME 254         524 FREE SECTORS

 A 004 DOS-KILLER
```

我們發現自由空間多出 524 － 492 ＝ 32 個磁區了！這是由第 1 、2 磁軌所多出來的 16 × 2 ＝ 32 個磁區。

本程式主要是將VTOC的第 1 、 2 磁軌的位元映射改成 FF（ 255 ），如此檔案便可使用這些空間了。第 160 行的 47084（$B7EC）就是 IOB 的第 4 位元組，存放的是WRTS所將讀取的磁軌號碼， 47085 則是磁區號碼，對 VTOC 而言其磁區位置是（17，0）。170 行將IOB的磁片號碼（第 3 位元組）設為 0，如此可以與任何編號的磁片符合，至於 POKE 47091 ，0 是設定為整磁區的讀寫。

第 180 行設定暫存區位址為 LOC＝10000 ，這個暫存區是用來存放用 RWTS 讀取的 VTOC 磁區，因此 LOC 要化成高低位址而存入 47088 及 47089 的 IOB 暫存區指位器，至此 IOB 已設

定完成。190 行是用來建立我們最主要的機械語言程式：

```
0300-    20 E3 03     JSR    $03E3
0303-    4C D9 03     JMP    $03D9
```

程式只有兩行，也就是由 $3E3 來取得 IOB 位址，然後 JMP RWTS。210 行設定RWTS動作狀態爲 1（READ），接著 CALL 768 進行讀取，RWTS 讀取到的是由 160 行所設定的（17，0）VTOC磁區，並且存放於 LOC。查VTOC的位元映射表（第二章）可以發現第 1、2 磁軌是對應於第60、61、64、65位元組，將 FF（255）POKE到這些位元組上，並由 230 行寫（呼叫碼＝2）回磁片 VTOC 位置，大功便告成了。這是一個很簡潔的RWTS 使用例，你可以發現並沒有很長的機械語言，更不需要更動太多 IOB 內含值。

RWTS使用例(II)：更改目錄表

```
]LIST

0   REM *******************************
1   REM **     FILE__LOCKER  ....     **
2   REM *******************************
100  POKE 768,32: POKE 769,227: POKE 770,
     3: POKE 771,76: POKE 772,217: POKE 7
     73,3
110 TRK = 17:RW = 1: POKE 47083,0: POKE 4
     7084,TRK: POKE 47091,0: POKE 47092,R
     W
```

```
120 LOC = 8192: POKE 47088,LOC -  INT (LO
    C / 256) * 256: POKE 47089, INT (LOC
     / 256)
140  TEXT : HOME : PRINT "LOCK OR UNLOCK?
     (U/L):";: GET L$: PRINT L$
149  REM *** READ A CATALOG SECTOR ***
150  FOR SEC = 15 TO 1 STEP  - 1: POKE 47
     085,SEC: POKE 47092,1: CALL 768
160  FOR X = LOC + 11 TO LOC + 221 STEP 3
     5:P0 =  PEEK (X):P2 =  PEEK (X + 2)
170  VTAB 10: HTAB 1: IF P0 = 0 THEN  POKE
     47092,2: CALL 768: GOTO 240
180  IF L$ = "L" THEN  POKE X + 2,P2 + 12
     8 * (P2 < 128): PRINT "LOCK:";
190  IF L$ = "U" THEN  POKE X + 2,P2 - 12
     8 * (P2 > 127): PRINT "UNLOCK:";
200  FOR J = X + 3 TO X + 32: PRINT  CHR$
     ( PEEK (J));: NEXT : PRINT
210  REM *** WRITE THE CATALOG SECTOR ***

220  NEXT X: POKE 47092,2: CALL 768
230  NEXT SEC
240  HOME : PRINT  CHR$ (4);"CATALOG"
```

程式 4 － 8　快速上鎖程式

這是RWTS的另一使用例*，我們可以一方面熟悉RWTS的用法，另方面來看看它的應用。與上一個程式不同的是這次所讀寫的是目錄磁區而不是VTOC，目錄磁區共有 15 個，由(17，15)至(17，11)，這便是 150 SEC 迴圈用 STEP －1 的原因，如此可以將目錄磁區順著次序一一讀出來。164 行的 LOC＋11 是因爲每個目錄磁區的 FDB 是由第 11 位元組才開始， FDB

---

* 程式 0 － 3 也是RWTS的BASIC程式使用例

長度爲 35 位元組，因而用 STEP 35 。 FDB 的第0位元組是 T／S表位置，若此位元組爲0，則表示目錄表至此結束，因而用P0 = PEEK (X)來取得第0位元組，並且以170行的 IF 來判斷目錄表是否已全部查完。 FDB 的第2位元組存放的是檔案類型及上鎖旗號（ b_7 ），若要上鎖則由180行將位元組值小於128的加上128，反之則由190行減128便可以了。

200行印出正被處理的檔案名稱共 30 個字元，220行將修改完畢的目錄磁區寫回磁片上，240行將結果以CATALOG展示出來。

直接使用RWTS時常會聽到喇叭發出聲音或螢幕不正常工作的情形，這是因爲 APPLE 監督程式用 $48 這個位址來存放狀態暫存器P的值，而RWTS卻使用（ $48,$49 ）來存於 IOB 位址，當由RWTS返回主程式時， $48 的內含值是 E8 ，其中的十進位旗號D（ b_3 ）被設成1，因而以 16 進位工作的監督程式就發生錯亂而造出怪異的現象，至於其他旗號則不受影響。解決的方法是在程式未結束（未交回監督程式）時將 $48 的內含值設爲0，程式4－8不歸0而能夠正常工作的原因是最後使用了 DOS 命令CATALOG，任何經由 FMS 的程式，在返回時，均會由 FMS 將 $48 的值置0，因而一切便是正常，而直接呼叫RWTS的程式，只好自己歸0。

4－7 RWTS的機械語言使用法

在機械語言當中使用RWTS，基本上與上一節的 BASIC 程式類似，只要設定好 IOB 然後呼叫RWTS，便可以自由的讀寫

磁片上任意指定的磁區了。為了能更確切體會 RWTS 的使用情形，我們以一連串的實驗來加以說明，首先請鍵入下列程式：

```
 2 ********************************
 3 **     RWTS      Utility ....    **
 4 ********************************
 5              ORG $300
 6 ***    RWTS entry    ***
 7 RWTS         EQU $3D9
 8 GETIOB       EQU $3E3
 9 USERERR      EPZ $0000
10 ***    Begin of Routine    ***
11 START        JSR GETIOB
12              JSR RWTS
13              BCC EXIT
14              STA USERERR
15 EXIT         LDA #$00
16              STA $48
17              RTS
18              END

*300.30E      （機械碼）

0300- 20 E3 03 20 D9 03 90 02
0308- 85 00 A9 00 85 48 60
```

程式 4－9　使用 RWTS的基本程式

這個程式由 11 行開始至 GETIOB （＄3E3）取得 IOB 起始位址，然後進入 RWTS，若 RWTS 無誤，則 C 旗號為 0；若否，則將錯誤碼寫入 USERERR，每次 RWTS 動作結束時要將 ＄48 歸 0，否則再次執行本程式後，喇叭會發出響聲。

修改VOL # —' DISK VOLUME'

使用 RWTS，我們可以讀寫磁片的任意磁區內容，譬如說，我們想修改 CATALOG時字幕最上頭的' DISK VOLUME'這幾個字，你可以將磁片上這些字所在的磁區用 RWTS 讀入暫存區，修改成我們所希望的字元，再用 RWTS 寫入磁片。如此一來，這張磁片的 DOS 永遠會在 CATALOG 的第一行顯示我們所要的字元了。

' DISK VOLUME ' 這些字是存在（ 02，02 ）磁區，因而我們要在 IOB 上設定磁軌、磁區值為：

```
*B7EB:00 02 02             ; IOB由B7E8開始
```

此處我們使用 DOS 原來的 IOB ，所以 B7EB 所含的是磁片號碼，設為 0 可以與任何磁片符合；而 B7EC 、 B7ED 就是 IOB 的磁軌、磁區存放位址，此處依我們的指定而設為（ 02，02 ）。

如果這時我們令 RWTS 做讀取動作，則（ 02，02 ）磁區的內容會被 RWTS 搬到 DOS 原來的暫存區內，為了避免破壞DOS 的暫存區資料，我們將 IOB 的暫存區位址指定在 $2000 。將 RWTS 的呼叫碼設定為 READ（ 01 ），插入實驗磁片：

```
*B7F0:00 20                ; 指定資料暫存區位址

*B7F4:01                   ; 設定呼叫碼

*300G
```

上圖中， IOB 的 $B7F0 存的是暫存區位址，$B7F4 存的是 RWTS 呼叫碼，當 300G 時，你會看到磁碟機動作了，若有任何不正常現象，則必然是手續上發生錯誤，請重新載入 DOS ，再行試驗。由於這些實驗直接將資料寫入磁片，操作不愼會將磁片內容毀壞，所以重新載入 DOS 時要使用完好的磁片，因爲實驗磁片的 DOS 可能已損壞了。

現在，(02 , 02) 磁區已讀入 $2000 了，請 20AFL :

```
20AF-   A0 C5      LDY   #$C5
20B1-   CD D5 CC   CMP   $CCD5
20B4-   CF         ???
20B5-   D6 A0      DEC   $A0,X
20B7-   CB         ???
20B8-   D3         ???
20B9-   C9 C4      CMP   #$C4
```

這些看似無意義的內容，就是 ' DISK VOLUME ' 這幾個字的 ASCII 碼，請修改如下：

```
*20AF:A0 D4 D3 C5 D4 A0 AD    ( TSET - )
```

請注意這些字元是倒著次序存放的。最後，將 RWTS 呼叫碼改成 WRITE (02) :

```
*B7F4:02

*300G
```

300G之後，請 PR#6 重新叫入 DOS ，接著：

```
]CATALOG

DISK - TEST 254

 A 004 DOS-KILLER
 A 004 FILE-LOCKER
```

剛才我們寫入在 $20AF 的字元出現了！請注意，上述過程是假定您均順利操作，若其間曾經執行任何 DOS IO（譬如 CATALOG），則 IOB 的內容會被 DOS 重新修定，因而您必須重新設定磁軌、磁區值（$B7EC、D）、暫存區位址（$B7F0、1）以及呼叫碼（$B7F4）。

修改CATALOG

現在再做一個修改，以更熟悉RWTS的操作：

```
*B7EC:01 0D        ;讀(01，0D)

*B7F0:00 20        ;至$2000

*B7F4:01           ;Read

*300G
```

這是將 DOS 與目錄列印相關的磁區（01，0D）讀入到$2000

，將 $2039 的內容修改後，寫回磁片：

```
*2039:4C DF BC        ; 修改DOS
*B7F4:02              ; Write
*300G
```

接著讀取（ 00 ，06 ）磁區的內容：

```
*B7EC:00 06           ; 讀（ 00 ，06 ）磁區
*B7F4:01              ; Read
*300G
```

修改成：

```
*20DF:20 0C FD C9 8D D0 03 4C 2C AE 4C 3C AE
```

最後寫入磁片：

```
*B7F4:02              ; Write
*300G
```

以 PR#6 重新載入 DOS 後，用一張長目錄的磁片來 CATALOG，這時如果您需要的檔名已找到，按下 RETURN ，目錄列印

的動作便結束了；若檔名尚未找到，按下其他鍵，便可以繼續列印下頁的檔名。

經過這些實驗，您已經對RWTS的使用有了相當程度的掌握了。無論如何，用手操作出來的體驗，大概是比任何複雜程式的觀摩更能給我們深刻的印象吧！

4－8 快速的資料搜尋法：DATA SEARCH

由上節以最直接的方法實驗了RWTS之後，現在來看在高階層次上RWTS有何用途。以高階層指令操作檔案之缺點是在於FMS 的作業過程是逐字讀取而不是段區（Block）搬動檔案資料，這種情況在資料尋找時最爲嚴重，因爲這些逐字讀取出來的資料還要用 INPUT 指令來存入Applesoft的字串，最後再由字串中找出所要的資料，在時間、空間上均很浪費。

其實資料一由RWTS讀入暫存區，其內容已經是字串的型式了（正 ASCII 碼），若我們能直接在暫存區內尋找所要的資料，當找不到時馬上令RWTS讀下一磁區，如此可省下 FMS 及 Applesoft 許多時間，速度上可以增快 4 至 20 倍呢！一個平常要5分鐘來尋找出資料的場合，我們大約 20 秒內就可以解決了。底下程式便是以RWTS由檔案中快速搜尋資料的例子：

```
2 ***********************************
3 *     Fast  Data  Search ....    *
4 ***********************************
5 *
6 SAVETS    EPZ $06
```

```
EOSFLAG   EPZ $07
STRLEN    EPZ $08
RECNUM    EPZ $19
RECLEN    EPZ $1C
NUMREC    EPZ $1E
TRECLEN   EPZ $F9
SAVEY     EPZ $FB
SAVEX     EPZ $FC
NXTS      EPZ $FD
RWTS      EQU $03D9
STRING    EQU $93F2
TSBUF     EQU $9400
DBUF      EQU $9500
FMWA      EQU $9800
          ORG $9300
          OBJ $800
***    INITIALIZATION    ***
          LDA FMWA
          STA NXTS
          LDA FMWA+1
          STA NXTS+1
          LDY #$0
          STY EOSFLAG
          STY RECNUM
          STY RECNUM+1
          STY SAVEY
          JSR NEWSEC1
          JSR SETRECL
***    START   OF   SEARCH   ***
          LDX #$0
          LDY SAVEY
START     LDA EOSFLAG
          BEQ BEGIN
          RTS
BEGIN     LDA DBUF,Y
          CMP STRING,X
```

```
             BEQ  MATCH
             LDA  STRING,X
             CMP  #$BF
             BEQ  MATCH
             LDX  #$00
             JSR  INCYR
             JMP  START
MATCH        JSR  INCYR
             INX
             CPX  STRLEN
             BNE  SKP3
             STY  SAVEY
             RTS
SKP3         JMP  START
***     ADJUST  COUNTERS    ***
INCYR        SEC
             LDA  TRECLEN
             SBC  #$01
             STA  TRECLEN
             LDA  TRECLEN+1
             SBC  #$00
             STA  TRECLEN+1
             BCS  SKP1
             INC  RECNUM
             BNE  SKP8
             INC  RECNUM+1
SKP8         SEC
             LDA  NUMREC
             SBC  #$01
             STA  NUMREC
             LDA  NUMREC+1
             SBC  #$00
             STA  NUMREC+1
             BCS  SKP2
             INC  EOSFLAG
             RTS
```

```
SKP2        JSR  SETRECL
SKP1        INY
            BNE  SKP4
            JSR  NEWSEC
SKP4        RTS
***      NEW  SECTOR  ROUTINE      ***
NEWSEC      LDY  SAVETS
            INY
            BNE  SKP6
NEWSEC1     LDA  NXTS
            BEQ  SKP7
            STA  TRACK
            LDA  NXTS+1
            STA  TRACK+1
            LDA  #TSBUF
            STA  BUFFER
            LDA  /TSBUF
            STA  BUFFER+1
            JSR  RWINIT
            LDA  TSBUF+1
            STA  NXTS
            LDA  TSBUF+2
            STA  NXTS+1
            LDY  #$0C
            LDA  #DBUF
            STA  BUFFER
            LDA  /DBUF
            STA  BUFFER+1
SKP6        LDA  TSBUF,Y
            BNE  SKP5
SKP7        INC  EOSFLAG
            RTS
SKP5        STA  TRACK
            INY
            LDA  TSBUF,Y
            STA  TRACK+1
```

```
            STX SAVEX
            STY SAVETS
            JSR RWINIT
            LDX SAVEX
            LDY SAVEX
            RTS
***   RESET RECORD LENGTH    ***
SETRECL     SEC
            LDA RECLEN
            SBC #$01
            STA TRECLEN
            LDA RECLEN+1
            SBC #$00
            STA TRECLEN+1
            RTS
***   CALL RWTS ROUTINE    ***
RWINIT      LDA /IOB
            LDY #IOB
            JSR RWTS
            RTS
***   IOB TABLE    ***
IOB         HEX 01600100
TRACK       HEX 1100FBB7
BUFFER      HEX 00950000
            HEX 0100006001
            BRK
            END
```

IOB

程式 4 － 10　以RWTS搜尋資料的程式

```
*9300.93F2

9300-  AD  00  98  85  FD  AD  01  98
9308-  85  FE  A0  00  84  07  84  19
9310-  84  1A  84  FB  20  7E  93  20
9318-  CB  93  A2  00  A4  FB  A5  07
9320-  F0  01  60  B9  00  95  DD  F2
9328-  93  F0  0F  BD  F2  93  C9  BF
9330-  F0  08  A2  00  20  48  93  4C
9338-  1E  93  20  48  93  E8  E4  08
9340-  D0  03  84  FB  60  4C  1E  93
9348-  38  A5  F9  E9  01  85  F9  A5
9350-  FA  E9  00  85  FA  B0  1B  E6
9358-  19  D0  02  E6  1A  38  A5  1E
9360-  E9  01  85  1E  A5  1F  E9  00
9368-  85  1F  B0  03  E6  07  60  20
9370-  CB  93  C8  D0  03  20  79  93
9378-  60  A4  06  C8  D0  2F  A5  FD
9380-  F0  30  8D  E5  93  A5  FE  8D
9388-  E6  93  A9  00  8D  E9  93  A9
9390-  94  8D  EA  93  20  D9  93  AD
9398-  01  94  85  FD  AD  02  94  85
93A0-  FE  A0  0C  A9  00  8D  E9  93
93A8-  A9  95  8D  EA  93  B9  00  94
93B0-  D0  03  E6  07  60  8D  E5  93
93B8-  C8  B9  00  94  8D  E6  93  86
93C0-  FC  84  06  20  D9  93  A6  FC
93C8-  A0  00  60  38  A5  1C  E9  01
93D0-  85  F9  A5  1D  E9  00  85  FA
93D8-  60  A9  93  A0  E1  20  D9  03
93E0-  60  01  60  01  00  11  00  FB
93E8-  B7  00  95  00  00  01  00  00
93F0-  60  01  00
```

程式 4 − 10 的機械碼

基本上這是一個尋找（search）副程式，它安排了兩個 256 位元組的暫存區來存放T／S及資料磁區，需要T／S磁區的原因是因爲本程式要逐一尋找檔案的各個磁區，因而需要T／S表來指示出檔案各磁區所在位置。程式中有 3 個零頁位址，分別用來計錄資料錄（Record）的長度（TRECLEN）、總數（NUMREC）及錄計數器（RECNUM）之用。另外還有變數 STRING 用以存放所要尋找的字串，STRLEN 則計錄字串長度（若超過 12 則程式要修改）。

程式一開始先設定各指位器的值，然後 JSR NEWSEC 將檔案的第一個T／S磁區及資料磁區分別讀入 TSBUF 及 DBUF。由於不使用 FMS 來處理檔案，因而T／S及資料磁區並不存放於 DOS 自己的暫存區，否則可能與 FMS 的流程衝突而造出錯誤的結果，資料讀入之後，便開始將每一個字元逐一比較。因爲 TRECLEN 是存放著錄的長度，所以每比較完一個字元，TRECLEN 便減一，一直到 TRECLEN = 0 時，便是找完一個錄了，這時 RECNUM（錄計數器最初爲 0）要加一而 NUMREC（總數）則減一。下一個錄開始時，TRECLEN 又恢復成錄的總長度，再繼續進行。因此，當我們找到所要資料時，只要以 POKE RECNUM 來找出該錄的號碼，而用 READ Filename, Rn 就可找出結果來了。當字串尋找時，如果您忘了字串的細節，可以用'？'號來取代不確定的字母，這種功能在即時性（Real time）的資料庫系統 * 相當有用。

本程式要由一個主程式來叫用，底下的 BASIC 程式便是一

* APPLE II BASIC資料庫系統設計，施威銘，波前 1984

個例子。程式首先會訊問資料所在的檔名、錄的長度、欄的個數、以及所欲尋找的字串，並且將這些資料POKE到程式4－10的各個對應位址（NUMREC、STRING等）上，然後CALL 37632（$9300）開始尋找，尋找結束之後，1110行以PEEK（7）來判斷字串是否找到，$7＝EOSFLAG是End Of Search旗號。字串所在的錄號碼是存於 RECNUM（$19,$1A）＝（25，26）位置上，PEEK出來後，便可以讀取了（1120行）。對於同一檔案的搜尋，各變數及旗號可以不必重新設定，因而1150行是CALL 37658（$931A，另一進入點）。

```
0  REM ******************************
1  REM **  FAST FILE SAERCH DEMO...  **
2  REM ******************************
3  REM
10  HIMEM: 37632:D$ =  CHR$ (13) +  CHR$ (4
    )
20  PRINT D$"BLOAD FAST FILE SEARCH.OBJ,A$9
    300"
100  REM ***  CALL THE ROUTINE  ***
110  GOSUB 1000: END
1000  REM ***  THE SUBROUTINE  ***
1010  HOME : PRINT D$"CLOSE"
1015  INPUT "FILE NAME: ";NA$: IF NA$ = "" THEN
     1010
1020  INPUT "RECORD LENGTH: ";RL$: IF RL$ =
     "" THEN 1020
1030  INPUT "NUMBER OF FIELDS: ";NF
1040  INPUT "SEARCH FOR: ";SS$:SS =  LEN (S
     S$): IF SS > 12 THEN 1040
1045  IF SS$ = "" THEN 1040
1050  FOR D = 1 TO SS:X = 120 +  ASC ( MID$
```

```
     (SS$,D,1)): POKE 37873 + D,X: NEXT : POKE
     8,SS
1060  PRINT D$;"OPEN ";NA$;",L";RL$
1070  POKE 28, PEEK (46525): POKE 29, PEEK
     (46526)
1080  PRINT D$"READ ";NA$;",R0": INPUT N
1090 NN =  INT (N / 256): POKE 30,N - NN *
     256: POKE 31,NN
1100  CALL 37632: REM *** ENTER SEARCH ***
1110  IF  PEEK (7) <  > 0 THEN  PRINT "END
     OF SEARCH": RETURN
1120 RN =  PEEK (25) +  PEEK (26) * 256: PRINT
     D$"READ ";NA$;",R";RN: FOR D = 1 TO NF
     : INPUT DA$: PRINT DA$: NEXT
1130  PRINT D$: PRINT "CONTINUE OR END ? (C
     /E): ";: GET G$: IF G$ = "E" THEN  RETURN

1140  IF G$ <  > "C" THEN 1130
1150  CALL 37658: REM   *** RENTER SEARCH *
     **
1160  GOTO 1110
```

*執行本程式前，請先以下列程式造出檔案

驗證上列程式前，請先造出檔案：

```
10  HOME : CLEAR :D$ =  CHR$ (13) +  CHR$ (4)
20 NR = 200:A$ = "FAST FILE ACCESS"
30  PRINT D$"OPEN TEST FILE,L200"
40  PRINT D$"WRITE TEST FILE,R0"
50  PRINT NR
60  FOR R = 1 TO NR
70  PRINT D$"WRITE TEST FILE,R";R
```

```
80  FOR F = 1 TO 3: PRINT A$;R: NEXT
90  NEXT R
100  PRINT D$;"CLOSE": END
```

由這個程式所造出的檔案共有201個錄，錄長度為200個字元，第0個錄字放檔案中錄的總數，其它則由 80 行來存入 3 組完全相同的' FAST FILE ACCESSn '字串，其中n由1～200。你可以用這組字串當中的任何部份（譬如CESS 96 、 ST FILE、ACCESS 41 等）來做為實驗時的搜尋字串。由本程式造出來的檔案共佔159個磁區，以一般方式來搜尋，費時約數分鐘（視字串位置而定），使用本程式則在數秒至數十秒之間。

執行列

```
FILE NAME: TEST FILE
RECORD LENGTH: 200
NUMBER OF FIELDS: 3
SEARCH FOR: ESS199
*最長約 35 秒
```

如果您使用' ? '來取代字母，則可以在找出第一組可能的字串，並且螢幕上印出：

```
CONTINUE OR END ? (C/E):
```

時，按下C，以便找出下一組可能的資料。

介紹了以RWTS讀寫VTOC及目錄表的方法，並且純手動的體會了RWTS的運作，最後還跳過DCI、FMS來與高階程式溝通，至此，RWTS的功能及其活用，相信我們都清楚的瞭解了吧！除了使用 DOS 模組（ FMS、RWTS ）之外，下一章我們將會看到如何修改 DOS 來造出有趣的功能，並且也要分析一下 DOS 在 APPLE 系統的地位及其溝通方式………

參考文獻：

Ref 1. What's Where in the Apple , W. F. Luebbrt , MICRO INK

Ref 2. Call APPLE All about DOS

Ref 3. Fast file search , A. E. Evenson , Nibble , Jan 1984

本章提要

DOS 的三大組成：DCI，FMS，RWTS，您有興趣進一步研究嗎？ DOS 如何與 BASIC 解譯器溝通？ DCI 的指令如何與 BASIC 指令配合？我們將介紹給您：

① DOS 如何攔截指令？如何過濾 BASIC 指令？

②各種向量（COUT、RESET、KBDIN）如何與 DOS 配合?

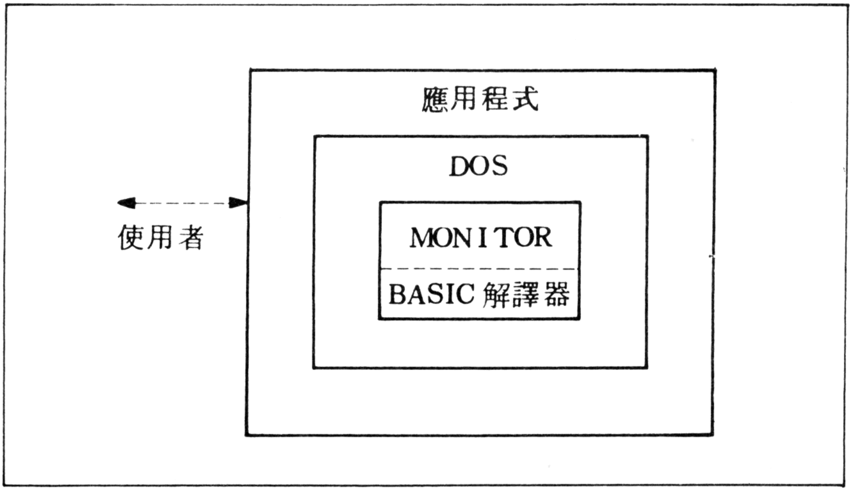
③ DOS 的簡易修改

④ DOS 的命令集如何修改？

⑤消去的檔案如何恢復？

⑥TYPE filename ?

5 DOS攔截：修改APPLE系統的方法



在第一、二册我們學習了RESET 、 COUT 等向量的用途，可是，裝上磁碟機之後，一切，就不對勁了……？

5－0 開機過程

APPLE 開機時，系統由POWER UP（$FAA6）開始來檢視磁碟機是否存在，若是，則JMP $CX00去起動磁碟機，當磁碟機起動、 DOS 載入主機之後，程式會 JMP $D43C 跳往 BASIC ROM 的主程式。BASIC 主程式平常所做的工作是掃描鍵盤，若發現按鍵輸入，則送往 BASIC 解譯器去分析命令內

容；若沒有按鍵，則回到 $D43C 做廻圈工作。

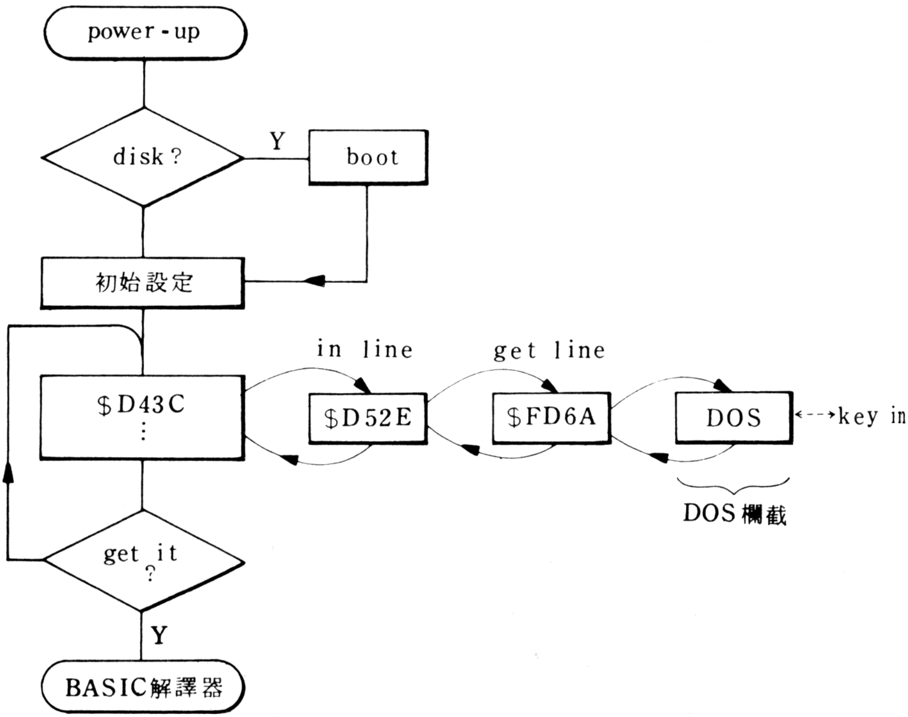


圖 5－0 APPLE 作業系統的主要流程

這樣的系統構造，令我們面臨了一個難題，那就是 APPLE 的監督程式及 BASIC 解譯器均是以 ROM IC 製成的，到底這種已定型的硬體架構要如何與後來才發展出來的 DOS 相互配合在一起？而加入 DOS 之後的 APPLE 在性質上又有怎樣的改變？爲何某些很有用的功能竟然走了樣呢？

5－1 DOS的I／O欄截

當初APPLE DOS是在BASIC 及Monitor ROM 之後設計的，因此如何將DOS 與已製成成品的 ROM 系統配合，確是一個問題。譬如，分別查看BASIC 及 DOS 手册，您會發現二者均有PR＃及 IN＃指令，但是其眞正的動作却不盡相同，請試著執行下列程式：

```
10 D$ =  CHR$ (13) +  CHR$ (4)
20  PR# 1
30  PRINT D$;"CATALOG"
40  INPUT "HIT RETURN TO RECONNECT
     DOS";A$
50  PRINT D$;"CATALOG"
60  END
```

程式5－0　沒有印字機的使用者，請將20行改成PR＃0

結果發現，雖然第 30 行及50 行均是 DOS 的CATALOG指令，但眞正在螢幕上看到的CATALOG 動作却只有一次（第 50 行），而第 30 行的CATALOG並沒有被當成DOS 指令來執行，事實上它是被當成一般的 PRINT 指令執行的，所以在' HIT RETURN……'這行字之上一行可以看到另外有一行'CATALOG'，這便是第 30 行所印出來的。

完全同樣的兩行程式，爲什麼竟然會有不同的結果呢？原來，APPLE DOS有一個巧妙的安排，那就是所有APPLE 的輸出

入均要由DOS先予檢查*，如果是DOS 指令（譬如是CATA-LOG）則DOS 便自己執行，如果是BASIC 指令則DOS 便將它傳給BASIC 解譯器來執行。可是 DOS 又如何能先檢查指令呢？由第一、二册我們都知道APPLE的I／O是由（$36，$37）及（$38，$39）這兩個指位器來指向的，如果我們令這兩個指位器指向DOS 而不直接指向輸出入口，那麼DOS 便可以當成APPLE 輸出入的關卡逐一的來檢查每一個I／O訊號了。

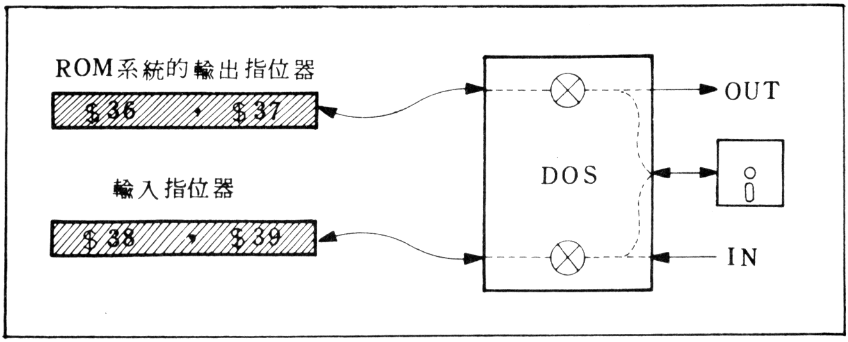


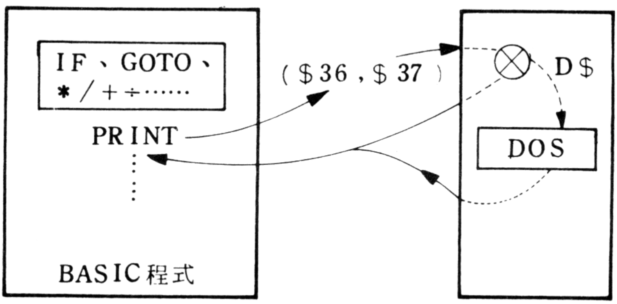
圖5－1 DOS欄截I／O訊號的情形

這便是所謂的DOS 欄截（Intercept）。就是由於所有I／O動作均由DOS 所欄截，所以我們鍵入CATALOG時 DOS 才會搶先來執行CATALOG 動作，並且不讓BASIC 解譯器發現，否則BASIC 解譯器會因爲不認識這個命令而在螢幕上印出SYNTAX ERROR呢！

現在問題便很清楚了！爲什麼DOS 指令均要使用PRINT來傳達，並且還要在前面加上D$（D$ 要事先定義爲CTRL-D）呢？這是因爲DOS 能夠欄截的只有I／O指令，但是當

* 圖5－0

BASIC 程式執行時，則只有 INPUT、GET、PRINT 等是屬於 I／O指令，並且 INPUT 及 GET 均會令程式停下來等候按鍵，所以可用以傳達 DOS 指令的便只有PRINT 了。



只有PRINT適合用來傳達DOS指令

但是PRINT 平常也要做印字的功能，爲了區分起見，凡是 PRINT 以D$（即CHR$(4)＝CTRL-D）爲開頭者均爲DOS 命令由DOS 來截取，而不以D$爲開始者則視爲一般的印字指令。

可是爲何程式5－0的 30 行也使用了D$ 爲開頭，却被當成普通印字指令呢？關鍵在第 20 行的PR#1，由第一、二册我們知道，PR# 指令是用來改變輸出指位器內含值的指令，一旦在BASIC 程式內使用了 PR#1，（$36，$37)就指向$C100 而不再是DOS 位址（$9EBD）了。一旦脫離了 DOS ，當然第 30 行的 PRINT 就被當成一般的印字指令而將D$與 CATALOG這些字元印出來，由於D$ 是控制碼不能由螢幕上看到，所顯示出來的只有′CATALOG′。但是 50 行的CATALOG 却爲何又能爲DOS 截取而執行呢？原因是雖然第 20 行改了輸

出指位器，但却沒有改掉輸入指位器，因此第 40 行的 INPUT 仍然會由 DOS 截取，而每次 DOS 截取指令之後，均會將輸出、入指位器重新設定成指向 DOS ，所以第 50 行是輸出指位器恢復後被執行的*，您可以將 PRINT PEEK(54)，PEEK(55)加入到程式 5－0 的第 25 及 45 行便可以看出輸出指位器變化的情形了。雖然此例的目的是為了說明 DOS 截取指令的情形，但這却也是使用 DOS 指令時易犯的錯誤，請多留意！

或許您會發現，如果直接由鍵盤鍵入 PR#1(或 0)指令，則 DOS 並沒有脫離，接下來的CATALOG 仍然可以正常動作，這是因為 DOS 截取了PR#1 指令，然後以 JSR($36)的方式送往印表機，由印表機周邊卡回來之後，控制權仍然屬於 DOS(因為 JSR 的關係)，這時 DOS 又將輸出指位器($36，$37)還原指向 DOS 了。至於程式 5－0 的PR#1 之所以能夠改變($36，$37)的值而令 DOS 脫離是因為程式執行中，控制權在 BASIC 解譯器，DOS 無法截取 PRINT、INPUT、GET 以外的指令，因此雖然 DOS 也有 PR#1 但這時却由 BASIC 的PR#1 來執行。你可以在程式 5－0 第 20 行的 PR#1 前加入D$，結果如何？至此，DOS 的欄截是否明白了？

底下是第一冊的程式 4－3，執行後請 RESET ，結果無法停。用 CTRL-C 停止程式後鍵入CATALOG結果是SYNTAX ERROR，其它的 DOS 命令均失去作用。CALL-151 3D0G 後，DOS 命令又恢復了，再執行一次程式，以CTRL-C 來暫停，這時 DOS 命令是會被接受的，但 RESET 後 DOS 又失

---

* 如果將 20 行改成 PR#0：IN#0，則輸出入指位器均會離開DOS，這時連第 50 行也不動作了。

去控制了！（爲何？）

```
0   REM  ***********************
1   REM  *   ON RESET GO .....  *
2   REM  ***********************
10   POKE 1010,102: POKE 1011,213
30   CALL  - 1169
40   INVERSE : PRINT "***-ABC-***"
50   GOTO 40
```

5－2 DOS攔截的徹底研究：KBDINCPT

首先，我們來看 DOS 對輸入的攔截。所謂的輸入共包含了①已開機狀態：包含開機後的鍵盤按鍵、程式執行時的 GET 與 INPUT 以及執行EXEC 檔案的讀取過程，這是因爲以EXEC 讀取的資料（命令）是被當成類如按鍵一般來執行的，所以也屬於熱開機。②冷開機狀態：當 DOS 第一次起動時，首先要做初始設定，然後才可進入已開機狀態。③由磁片讀取檔案資料。這三種狀況在 DOS 輸入攔截過程中必須分別加以考慮。

對於像 DOS 這種複雜程式的解析，我們必須適當的加以分割才有辦法清楚理解，譬如本節所要說明的KBDINCPT，其實也牽涉到極大部份的 DOS 程式，但由程式5－1 可以看出我們已集中了其重要部份，而對於較爲旁支的副程式（第 7～10 行）則均止於其標名及進入位址，這並不意味著我們將不去瞭解這些副程式，而是爲了將討論焦點集中的緣故。

輸入攔截程式：KBDINCPT

```
!L
         ;
         ********************************
         * DOS keyboard intercepter *
         ********************************
         *
CSW      EPZ $36
KSW      EPZ $38
CH       EPZ $24              ;CURSOR COULUMN
BASL     EPZ $28              ;LEFT MARGINE OF CURRENT LINE
INIT1    EQU $9DEA            ;PROCESSING AFTER DOS COLDSTART
VIDINCPT EQU $9EBD            ;DOS OUTPUT (video) INTERCEPT
FILEREAD EQU $A626            ;READ DATA BYTE FROM FILE
EXECREAD EQU $A682            ;READ BYTE FROM EXEC FILE
         *---------------------------------
         * RELOCATEABLE ADDRESS CONSTANTS:
         *---------------------------------
         ORG $9D02
DKBDINCP ADR KBDINCPT
DVIDINCP ADR VIDINCPT
         *---------------------------------
         * KEYBOARD INTERCEPT ROUTINE :
         *---------------------------------
         ORG $9E81
KBDINCPT JSR SETUP
         *
         * TEST STATUS :
         *
         LDA STATFLG
         BEQ KBD2
         *
         * READ FROM FILE OR COLDSTART:
         *
```

```
         PHA                  ; 暫存 STATFLG
         LDA ASAV
         STA (BASL),Y         ;REMOVE CURSOR FROM SCREEN
         PLA                  ; 取回 STATFLG
         BMI KBD1             ;COLDSTART
         JMP FILEREAD         ;READ FILE BYTE
*
* COLDSTART:
*
KBD1     JSR INIT1            ;1ST ENTRY PROCESSING
         LDY CH
         LDA #$60             ; 閃爍方塊
         STA (BASL),Y         ;REPLACE CURSOR ON SCREEN
*
* WARM START:
*
KBD2     LDA EXECFLG
         BEQ KBD3             ;NOT 'EXEC'
         JSR EXECREAD         ;READ 'EXEC' FILE BYTE
KBD3     LDA #3               ;SET STATE 3 OUTPUT HANDLER
         STA CSWSTATE         ; ECHO INPUT ON SCREEN
         JSR REGRST           ;RESTORE REGISTERS
         JSR KSWGO
         STA ASAV
         STX XSAV
         JMP DOSEXIT
KSWGO    JMP (KSW)
*------------------------------
* INTERCEPT SETUP SUBROUTINE :
*------------------------------
         ORG $9ED1
*
* SAVE REGISTERS:
*
SETUP    STA ASAV
         STX XSAV
```

```
         STY YSAV
         TSX
         INX             ；堆疊指位器往下移二格
         INX
         STX SSAV        ；令S暫存器指向DOS進入點
*
* RESTORE TRUE I/O HANDLERS TO I/O HOOKS:
*
         LDX #3          ；搬動4個位元組
IOTRUE   LDA CSWTRUE,X   ；將$FD1B及$FDF0
         STA CSW,X       ；存入CSW及KSW
         DEX
         BPL IOTRUE
         RTS
*------------------------------
* EXIT SUBROUTINE :
*------------------------------
         ORG $9FB3
*
* RESTORE DOS INTERCEPTS:
*
DOSEXIT  JSR IOINIT
*
* RESTORE REGISTERS:
*
         LDX SSAV
         TXS
REGRST   LDA ASAV
         LDY YSAV
         LDX XSAV
         SEC
         RTS
*------------------------------
* INTERCEPT RESTORE SUBROUTINE:
*------------------------------
         ORG $A851
```

```
*
* DOS KEYBOARD VECTOR STILL SET:
*
IOINIT   LDA KSW+1
         CMP DKBDINCP+1
         BEQ IOINT1
*
* RESTORE DOS KEYBOARD VECTOR:      ; 還原輸入指位器
*
         STA KSWTRUE+1
         LDA KSW
         STA KSWTRUE
         LDA DKBDINCP
         STA KSW
         LDA DKBDINCP+1
         STA KSW+1
*
* DOS VIDEO VECTOR STILL SET:
*
IOINT1   LDA CSW+1
         CMP DVIDINCP+1
         BEQ IOINT2
*
* RESTORE DOS VIDEO VECTOR:         ; 還原輸出指位器
*
         STA CSWTRUE+1
         LDA CSW
         STA CSWTRUE
         LDA DVIDINCP
         STA CSW
         LDA DVIDINCP+1
         STA CSW+1
IOINT2   RTS
*----------------------------
* MAIN ROUTINE VARIABLES :
*----------------------------
```

```
141          ORG $AA51
142 STATFLG  DFS 1
143 ;$0=WARMSTART,  $01=READ
144 ;$40=RAM APSOFT, $80=COLDSTART
145 CSWSTATE DFS 1              ;CSW INTERCEPT STATE
146 CSWTRUE  DFS 2              ;                     眞正的輸出程式(通常爲$FDF0)
147 KSWTRUE  DFS 2              ;TRUE INPUT  HANDLER  眞正的輸入程式(通常爲$FDIB)
148 *
149          ORG $AA59
150 SSAV     DFS 1              ;SAVE S_REG
151 XSAV     DFS 1              ;SAVE X_REG
152 YSAV     DFS 1              ;SAVE Y_REG
153 ASAV     DFS 1              ;SAVE A_REG
154 *
155          ORG $AAB3
156 *
157 EXECFLG  DFS 1              ;EXEC FILE FLAG (0=INACTIVE)
158          END
```

程式5－1　DOS的輸入攔截程式KBDINCPT

程式5－1的INIT 1是DOS 冷啓動時的初始設定副程式，VIDINCPT（$9EBD）是輸出攔截程式，FILEREAD是檔案讀取副程式，其功能是將目前所開啓的檔案讀取一個位元組並存於A暫存器，若檔案已結束則關閉檔案並印出′END OF DATA′。EXECREAD也是由檔案讀取一個位元組的副程式，但其所讀的檔案是自動檔（可以用EXEC執行的檔案）。

本程式稱爲DOS 的輸入攔截程式，其進入點由 24 行開始，第一個動作是JSR SETUP，67 行至 82 行的SETUP 除了暫存所有的暫存器值之外，最重要的便是將輸出入指位器指向原

來（眞正的）監督程式的I／O處理程式（KEYIN＝$FD1B及COUT 1＝$FDF0），這時若發生程式中斷，則DOS 便失去作用了。將I／O權暫時交回之後，DOS便開始檢查狀態旗號(STATFLG），狀態旗號分別由 00 代表熱開機、01 代表讀檔案、$40代表RAM BASIC *、$80代表冷開機，28 行判斷旗號若爲0則至 49 行的KBD2 做熱開機工作（即正常的按鍵讀取工作），37 行再次檢查旗號（旗號值由 28 行 LDA 之後，於 33 行PHA ，再由 36 行 PLA 取回），若爲 $80 （負值）則至 42 行的KBD1 做冷開機，否則，跳至FILEREAD讀取一個位元組。

由以上的動作分析，我們可以發現，一旦發生輸入動作，監督程式的RDKEY（$FD0C），經由JMP（KSWL）便跳往DOS的KBDINCPT ，DOS 欄截之後，馬上由SETUP 將輸出入控制權交給眞正的I／O程式，然後分成3條路線來進行。如果是程式進行時的檔案讀取，則直接跳至FILEREAD ；若是按鍵讀取（熱開機），則先判斷是否正在執行EXEC命令，若是，則要JSR EXECREAD ，否則由 53 行設定輸出狀態（CSWSTATE）爲3（輸出至螢幕），由 54 行取回暫存器值，並且至眞正的讀鍵副程式（$FD1B，由SETUP設定）讀取一個字鍵（第 55 行使用JSR KSWGO ，因此主權仍由DOS 掌握），最後由DOSEXIT結束。DOSEXIT除了取回各暫存器值之外，最重要的是做IOINIT（108行至137行），IOINIT將當初由SETUP所設定的眞正I／O（CSW 及 KSW ）程式位址存回KSWTRUE及CSWTRUE，然後將CSW 及KSW 又重新設定成DVIDINCP及DKBDINCP ，因此輸出入權又由DOS 所掌握

* 早期的BASIC系統。

了。這便是爲何程式 5 － 0 第 40 行的 INPUT 可以恢復 DOS 命令的原因了，因爲雖然 PR#N 令 DOS 輸出攔截失效，但輸入攔截仍然有效，所以 INPUT 一呼叫 RDKEY ，RDKEY 於是 JMP（ KSW ），也就跳至 KBDINCPT， 當結束 KBDINCPT 時 DOSEXIT 連 CSW 也還原成 DOS 的 VIDINCPT（即 DVIDINCP ）了，所以程式 5 － 0 第 50 行才能以 PRINT 來傳達 DOS 命令。

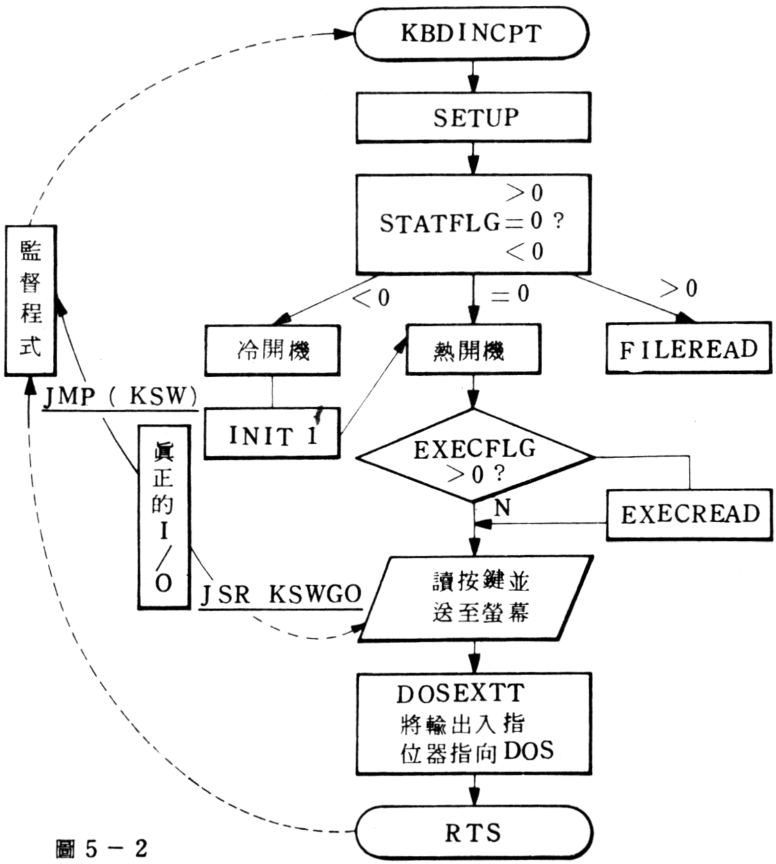


圖 5 － 2

DOS 的輸出攔截也一樣使用了 DOSEXIT 副程式，因此也一樣會將輸出入指位器還原，這便是爲何每次我們將（$38，$39）的值改變後（CALL-151 [RETURN] 38：00 03 [RETURN]），再次查看其值竟然又是 38：81 9E 的原因了。因爲修改完畢 RETURN 後，游標重現時，便呼叫了 COUT（參考 Ref 2），COUT JMP（CSW）後便爲 DOS 所攔截了，而 DOSEXIT 後（$38，$39）的值當然又還原了。

譬如說，我們設計了底下的程式：

```
0800          ****************************************
0800          *   CTRL-C--------------> CATALOG  *
0800          ****************************************
FD1B          KEY_IN   EQU $FD1B
A56E          CATALOG  EQU $A56E
03D0          W_START  EQU $3D0
0300                   ORG $300
0300          *
0300 20 1B FD          JSR KEY_IN          ;GET KEY
0303 C9 83             CMP #$83            ;CTRL-C?
0305 D0 06             BNE DONE
0307 20 6E A5          JSR CATALOG
030A 20 D0 03          JSR W_START
030D 60       DONE     RTS
030E                   END
```

程式 5－2　以 CTRL-C 來代替 CATALOG 的程式

這個程式可以在我們鍵入 CTRL-C 時，執行 CATALOG 的動作，我們希望每次按鍵時，APPLE 會自動的來執行這個程式，解決的方法是將輸入指位器指向本程式的進入點（$300），但是由前述的理由，我們知道由監督程式中鍵入 38：00 03 是不會

成功的，檢視程式 5 － 1 我們發現有幾處進入點可以更改輸入指位器：

①IOINIT ：第 108 行的 IOINIT （ ＄A851 ）雖然是令輸出入指位器指向 DOS ，但它會先將目前的KSW及CSW 值取入KSWTRUE、CSWTRUE，如果在監督程式鍵入 38：00 03 馬上執行 IOINIT，則KSWTRUE就指向＄300 了，而一旦DOS接管了輸出入，KSW 又被設成KSWTRUE＝＄300，大功便告成了。

```
CALL-151 [RETURN]
300 : 20 1B FD C9 83 D0 06 20 6E A5
      20 D0 03 60 [RETURN]
38 : 00 03 NA851G [RETURN]
```

＊上一行的N是為了避免A851被監督程式當成數值而輸入到 3A、3B位址內

②DOSHOOK：在第 3 頁的 DOS 向量當中，＄3EA 是稱為DOSHOOK，它跳向 IOINIT，因此我們也可以用 38：00 03 N3EAG 來達成同樣的目的。

③KSWTRUE：我們可以更直接的來修改KSWTRUE＝＄AA55的內含值，AA55：00 03 結果也是一樣的。不過，所有這些修改，當按下RESET 之後又都會被 DOS 還原了。

另外，程式 5 － 1的 77 行至 82 行這段程式是 DOS 用來令輸

出入指位器指向眞正輸出入副程式用的，如果我們直接執行這個程式，DOS 將會完全與 APPLE 系統脫離（＊ 9EE0G 或 CALL 40672 ），在某些暫時不用 DOS 的場合，這個方法很有用，並且只要 [RESET] 之後，DOS 又會連接回來了。

```
插入主磁片，開機
CALL 40672
CATALOG 結果 SYNTAX ERROR ?
[RESET] 之後 CATALOG …………
```

明白了上述的原理，一切與輸出入相關的向量就可運用自如，而不再受 DOS 的困擾了吧？

5－3 DOS的修改

除了以上一節的欄截方式來修改 APPLE 功能之外，我們也可以直接修改 DOS 來造出各種特殊效果。如果將修改後的DOS 用 INIT 或第四章的 PATCHING DOS程式來寫入磁片的話，則每次 DOS 載入時，其特殊功能也跟著動作，如此可以免除操作上的麻煩，對於應用程式的場合，相當有用。本節選錄了各種簡短實用的方法，列出如下：

印出被消去的檔名

```
*ADD9:EA EA      ; 修改(1)
*ADD9:30 4A      ; 恢復
*ADD9:10         ; 修改(2)
```

經由修改(1)可以將被 DELETE 的檔名在 CATALOG 時一併列出來，其中已被消除的檔名最後字元（在螢幕右方）會呈現一個反白字形。ADD9 這個位址位於 FMS 當中，其作用是檢查目錄檔名是否已消去，經我們設爲 NOP 之後，就失去作用而將全部檔名印了出來。不過若消去的檔名已受新檔名的覆蓋，則便無法印出來了。

ADD9 的原指令是以 BMI 來檢查檔案是否被註銷（ FDB 的首位元組爲 FF ），若將 BMI 改成 BPL ，則 CATALOG 時便僅印出被消去的檔名，見修改(2)。鍵入 ADD9 ： 30 4A 便可恢復 DOS 的原程式。

直接執行 OPEN……

像 OPEN、READ、WRITE、……這些指令通常是必須由 BASIC 程式執行，可是在某些場合，我們會希望經由直接命令來加以執行，只要鍵入：

```
*A021:EA EA EA   ; 修改
*A021:20 5E A6   ; 恢復
```

便可達到目的了。A021 位於 DCI 內部，用以檢查APPLE 是否處於執行狀態，將這個動作去除，OPEN、READ、……等便可以直接執行了。

增加BSAVE長度

如果您嘗試過BSAVE TEST，A$800，L$BFFF ，結果會是RANGE ERROR 。做下列修改，你便可以存更長的B 型檔：

```
*A964:FF          ；修改
*A964:7F          ；恢復
```

執行各種檔型的HELLO

通常HELLO 程式必須是A 型檔，請將目前的HELLO 消除，然後BSAVE HELLO ，A$800，L$20，重新開機後，你會得到一個TYPE MISMATCH的錯誤訊息。做下列修改，便可以執行B 型或T 型的HELLO 程式：

```
*9E42:34          ；B型檔
*9E42:14          ；可以EXEC的T型檔
*9E42:06          ；恢復
```

維持MON狀態

做程式偵錯時，我們往往會用MON C，I，O 來幫助追踪磁碟作業過程，在發現錯誤時我們會按下 RESET 令程式停止，修改後再繼續偵錯。可是這時 MON 的功能便失去了，而須再行鍵入，下列的修改可以令 MON 功能在 RESET 之後仍舊保存：

```
*9DD7:EA EA          ；修改
*9DD7:8D 5E          ；恢復
```

縮寫的檔名

APPLE 所附的 FID 程式，可以用 ' = ' 號來縮寫檔名，譬如＝.ABC 表示所有以 .ABC結尾的檔名。使用下列修改，您也可以令 DOS 擁有這樣的能力（我們以 ' — ' 號來做縮寫）：

```
;
********************************
*    A WILDCARD PATCH .....    *
********************************
PTR        EPZ $42
DIRENTRY   EQU $B4C6
DIRINDEX   EQU $B39C
NOTFOUND   EQU $B20B
           ORG $BA69
COMPARE    INX
           LDA (PTR),Y
```

```
12                CMP  DIRENTRY,X ;比較每個字元
13                BNE  WILDCARD   ;若發現不同，至WILDCARD處理
14                INY             ;（$A458）
15                CPY  #$1E       ;檔名30個字元已結束？
16                BNE  COMPARE    ;否，繼續
17  OK            LDX  DIRINDEX   ;OK，找到檔案名稱
18                CLC
19                RTS
20  WILDCARD      CMP  #$AD       ;若是'—'之前完全相同，OK
21                BEQ  OK
22                JMP  NOTFOUND   ;否則比較下一檔案名稱
23                END

*BA69.BA81

BA69- E8 B1 42 DD C6 B4 D0
BA70- 0A C8 C0 1E D0 F3 AE 9C
BA78- B3 18 60 C9 AD F0 F7 4C
BA80- 0B B2
*
```

```
*B201:   4C 71 BA        ;(JMP $BA71)
```

這個程式是由 DOS 檔名比較程式中跳出（JMP $A458），然後做縮寫處理。這個程式其實只在 DOS 中增加了第 20 及 21 行而已，不過因爲該處空間已滿，只好借用 $BA69 的這個 DOS 空區。第 10 至 13 行是檔案名稱的比較，若發現不同，則往 NOTFOUND 去比較下一個檔名。我們加入的 20 及 21 行便是攔截住往 NOTFOUND 的通道，首先檢查是否因爲縮寫符號'－'而造成的 BNE（13 行），若是，則送往 OK，當成是找到檔案名稱了，若否，才送往 NOTFOUND。

這個修改不長，可以直接鍵入。試驗時，請注意先 BLOAD

程式的機械碼後，再修改B201 的值，否則B201 一 修改，DOS 找檔案時會跳往 BA71 ，但這時程式尚未載入，系統就Down 了。

D$＝#

每次以PRINT 來使用 DOS 命令時，均要先附加上 D$，當然也要先定義D$ = CHR$(4)，下列的修改可以令您設定其他任意字元來呼叫 DOS 命令：

```
]POKE 43698,128 + ASC ("#")          ；修改
]POKE 43698,132                      ；還原
```

經過上述修改後，你就可以用 BASIC 命令：PRINT “ # ××××……”來叫用 DOS 命令了。

DOS-KILLER

上一章介紹的DOS－KILLER 程式是用來將磁片上 DOS 所佔用的空間讓出來存放檔案資料，這個程式對於已存有資料的磁片很有用。可是對於一片空白磁片，我們可以在 INIT時就做好這個工作：

```
]POKE 42344,76   ⎫
]POKE 44723,4    ⎬ 修改
]POKE 46922,96   ⎭
```

```
]POKE 42344,32
]POKE 44723,12      } 還原
]POKE 46922,173
```

起動 DOS ，做上述的修改然後INIT 空白磁片，這時 INIT的磁片不但不含有 DOS ，並且連 HELLO 程式也不存入，用這樣的磁片來存DATA是最適當不過了。

DISK VOLUME 0

```
]POKE 42332,0        ；修改
]POKE 42332,254      ；還原
```

雖然APPLE 的 INIT可以設定磁片號碼，但卻無法將號碼設成 0 。號碼爲 0 的磁片有許多用途，因爲它可以適合其它號碼，上述的修改就可以令您INIT 出號碼爲 0 的磁片，請 POKE 後用空白磁片 INIT 看看。

5－4 消除掉的檔案如何恢復

在上一節的 DOS 修改當中，我們曾經將已消除的檔名印了出來，可是每個檔案最後的反白字似乎不全相同，它們代表了什麼意義呢？在第二章當中，我們提到目錄表內 FDB 的最前面兩個位元組代表的是 T／S 表的起始磁區位置，而在該檔案被消除時，最前面的位元組會被設定成FF ，並且其原來存放的磁軌號

碼會被寫入到檔名區的最後位元組上去。

這是說，雖然檔案被消除了，其實只不過是將FDB 的開始位元組做個記號（FF ）而已，並且其原來資料也沒有損壞。因爲在檔名的最後位元組上我們可以找回被FF 覆蓋住的T／S表磁軌號碼，因此，只要原檔案不被新檔案覆蓋，要將已消除的原檔案恢復並不是太困難的事情。所以，請記住，萬一不小心誤將檔案 DELETE 時，千萬不要在磁碟機運轉當中按下 RESET ，因爲這樣做可能令VTOC 表的更改過程發生錯誤，而將整張磁片破壞，並且，更可怕的是，這種破壞表面上可能不易看出，可是却會令重要的資料錯亂失散，造成無法彌補的損失。務必等到 DELETE 動作結束後，再依我們的方法來處理，一切便很安全了。

恢復檔案的方法很多，最直接的方法是利用3－0節的方法查出目錄區，用ZAP 將檔名最後字元恢復成空格，將T／S表的磁軌號碼抄回FDB 的第一位元組，最後用FID 來恢復檔案。不過每次使用ZAP 及修改磁區均是麻煩的事情，底下是一個很方便使用的程式，全部以BASIC 寫成，程式運作極爲清晰明瞭：

```
1  REM ******************************
2  REM *  DELETE  &  RELETE  FILES  *
3  REM ******************************
5  ONERR  GOTO 190
10 D$ =  CHR$ (4): HOME
15 F$(1) = "DELETE":F$(2) = "RELETE"
20  PRINT "DO YOU WISH TO": PRINT : PRINT
    "1)"F$(1): PRINT  TAB( 17)"OR": PRINT
```

```
     "2)"F$(2): PRINT : PRINT  TAB( 12)"F
     ILES? 1) OR 2)";: INPUT A$
50 ME =  VAL ( LEFT$ (A$,1)): IF ME < 1 OR
     ME > 2 THEN  END
60 DO = 256 *  PEEK (978):XO = DO + 4139
70  ON ME = 2 GOSUB 160
80  PRINT : PRINT "FILE NAME ?": INPUT "-
     -->";A$: IF A$ = "" THEN  GOSUB 120:
      GOTO 80
90  PRINT D$F$(ME)A$                          主程式
100  POKE 216,0: INPUT "NEXT ONE (Y/N) ?"
     ;A$: ON  LEFT$ (A$,1) = "Y" GOTO 80
110  ON ME = 2 GOSUB 190: END
120  REM  CATALOG FILES  印出目錄的副程式
130  IF ME = 2 THEN  POKE XO + 174,16
140  PRINT D$"CATALOG": POKE XO + 174,48:
      RETURN
150  REM  RELETE MODE     恢復檔案
160  POKE XO + 18,230: POKE XO + 27,160: POKE
     XO + 29,230: POKE XO + 1224,16: POKE
     XO + 1240,29: POKE XO - 388,230: POKE
     XO - 1171,82
170  POKE XO + 24,198: POKE XO + 1482,89
180  RETURN
190  REM  DELETE MODE     消去檔案
200  POKE XO + 18,198: POKE XO + 27,255: POKE
     XO + 29,198: POKE XO + 1224,48: POKE
     XO + 1240,30: POKE XO - 388,198: POKE
     XO - 1171,68
210  POKE XO + 24,230: POKE XO + 1482,25
215  IF  PEEK (222) = 6 THEN  END
220  RETURN
```

程式 5－3　恢復被消除的檔案

本程式共分成兩部份。一部份是DELETE ，其動作與正常 DOS 命令無異。另一部份是與 DELETE 相反的 RELETE，RELETE 並不是 DOS 命令，因此要用POKE 43160，82將 DOS 的DELETE 名稱改成RELETE ，這時DOS 便只認得 RELETE 而不承認 DELETE了。如果當程式執行 RELETE 當中按下 RESET ，然後使用DELETE 指令，結果會是 SYNTAX ERROR。

雖然已將命令名稱改過，但是DOS 做的仍舊是 DELETE 的工作，因此對於 DELETE程式本身也要加以修改才行。DELETE 動作的第一步是由目錄表當中找出檔名進入點（FDB），因此勢必比較各檔名的每一字元，對於一個已被消除的檔名而言，其最後字元因存放了T／S 表所在的磁軌號碼，因而會被當成相異檔名不被找出，對此，我們可以令其只比較前 29 個字元，檔名便可以找出來了，所以POKE 45571，29 。

另外，由於DELETE 是將FDB 的第一位元組內含值的T／S 磁軌號碼搬入第 33 位元組，而RELETE 則反之，因此要將$AD42 的STA $B4E6，X改成STA $B4C6，X ，所以 POKE 44355，198 。同時，也要將DELETE 在 FDB 第一位元組存入FF 的動作改成在第 33 位元組填入空白，因此 POKE 44360，230：POKE 44358，160， 160是空格的 ASCII 碼。當檔案開啓時，檔案的T／S 表磁軌號碼要存入檔案工作區，但是在RELETE 尚未恢復 FDB 前，檔案已打開了，所以要由檔名的第 30 位元組取磁軌號碼，因此 POKE 43943，230 來將檔案開啓副程式的ABA6：LDA B4C6，X改成LDA B4E6，X 。

最後，也是最重要更改的部份是DELETE 的 VTOC更新動作。由於DELETE 時，會將檔案T／S 表上所記錄的磁區在位元映射表上一一設爲 1，所以RELETE 要將它改成0，這只要將B2F5 的ORA BTMPEND，Y 改成EOR BTMPEND，Y 就可以了，所以POKE 45813，89 。程式中以X0＝256＊PEEK（978 ）＋ 4139 來做位址計算的基準，如此可以適應各種記憶量的場合，其中978＝＄3D2存的是DOS 起始頁數，通常爲＄9D 。

5－5 修改DOS的命令集：自動修改程式

像CATALOG這種命令，因爲常常使用，有時也不免覺得太長而希望以 CAT 或其它較短的名稱來代替；而當您設計一套實用軟體時，更會希望有較合宜的名稱來代替 DOS 命令，一方面使操作者易於接受，另方面也有程式保密的作用吧？此處，我們介紹您一個可以用來修改 DOS 命令名稱的程式，除了更改名稱之外，還會顯示目前所用的 DOS 命令集，如果您拿到一套經過別人更改命令名稱的程式時，這種命令名稱的顯示功能就很有用了。

DOS命令修改程式

```
0   REM ******************************
1   REM *   DOS COMMAND CHANGER ..   *
2   REM *   HAVE FUN ..............   *
```

```
3  REM ****************************
4  REM
20  TEXT : HOME
30  GOSUB 2010: GOSUB 610
100  REM  PEEK COMMAND TABLE & CREATE ARR
     AY
102  REM  ARRAY DOS$(R1-28,C1-2)  C1=COMM
     AND  C2=START ADDR
104  REM  DOS$=DOS COMMAND TABLE
106  REM  DOS=ADDR COMMAND TABLE
110 TM = START       : REM 命令表由 43140 開始
120  IF  PEEK (TM) = 0 THEN FIN = TM - 1:
      GOTO 140: REM  FIND END OF TABLE
130 TM = TM + 1: GOTO 120
140 I = 1: FOR J = 1 TO 29: FOR K = 1 TO
     2:DOS$(J,K) = "": NEXT K,J:DOS$(1,2)
      = STR$ (START):DOS$ = "": REM INIT
     IALIZE
150  FOR DOS = START TO FIN
160  IF  ASC ( CHR$ ( PEEK (DOS))) > 127 THEN
     DOS$(I,1) = DOS$(I,1) +  CHR$ ( PEEK
     (DOS)):DOS$ = DOS$ +  CHR$ ( PEEK (D
     OS)):DOS$((I + 1),2) =  STR$ (DOS +
     1):I = I + 1: GOTO 180: REM IF HI BY
     TE INCR I
170 DOS$(I,1) = DOS$(I,1) +  CHR$ ( PEEK
     (DOS)):DOS$ = DOS$ +  CHR$ ( PEEK (D
     OS))
180  NEXT DOS: RETURN
300  REM  *** DEC ---> HEX ***
310 HD% = DOS / 256:NBR = HD%: GOSUB 340:
     HB$ = HEX$
320 LD% =  FN MOD(DOS):NBR = LD%: GOSUB 3
     40:LB$ = HEX$
330 HEX$ = HB$ + LB$: RETURN
340 H% = NBR / 16 + 1:L% = NBR / 16:L = L
     % * 16:L% = NBR - L + 1
350 HEX$ =  MID$ (H$,H%,1) +  MID$ (H$,L%
```

```
     ,1): RETURN
400  REM  REORGANIZE COMMAND TABLE
410  IF PT = 1 THEN T1$ = "": GOTO 430
420 T1$ =  LEFT$ (DOS$, VAL (DOS$(PT,2)) -
     START)
430  FOR I = 1 TO  LEN (NC$`:T2$ = T2$ +
      MID$ (NC$,I,1): NEXT
440  IF PT = 28 THEN T3$ = "": GOTO 460
450 T3$ =  RIGHT$ (DOS$,FIN + 1 -  VAL (D
     OS$((PT + 1),2)))
460 DOS$ = T1$ + T2$ + T3$:T2$ = ""
470 DOS = START
480  FOR I = 1 TO  LEN (DOS$): POKE DOS, ASC
     ( MID$ (DOS$,I,1)):DOS = DOS + 1: NEXT

490 FIN = FIN +  LEN (NC$) -  LEN (OC$)
500  POKE FIN + 1,0: RETURN
600  REM *****  MENU  *****
610  HOME :TT$ = "===============": GOSUB
     3110
620 TT$ = "TRICK DOS MENU": GOSUB 3110
630 TT$ = "===============": GOSUB 3110
640  VTAB 6: PRINT "1.DISPLAY CURRENT DOS
      COMMAND TABLE.": PRINT
650  PRINT "2.CHANGE DOS COMMAND TABLE.":
      PRINT
680  PRINT "3.EXIT PROGRAM.": PRINT : PRINT

690  VTAB 17: CALL  - 958: PRINT "     WHI
     CH CHOICE? ";: GET I$: PRINT I$:CH =
      VAL (I$)
700  IF CH < 1 OR CH > 3 OR I$ = "" THEN
     690
705  IF CH = 3 THEN  END
710  ON CH GOTO 800,1000
800  REM  ***DISPLAY CURRENT TABLE***
810  HOME :TT$ = "=========================
     =========": GOSUB 3110
820 TT$ = "CURRENT DOS COMMANDS & ADDRESS
```

```
     ES": GOSUB 3110
830 TT$ = "=================================
     ==": GOSUB 3110
840  IF  NOT FF THEN  VTAB 8: INVERSE :TT
     $ = " READING DOS COMMAND TABLE ": GOSUB
     3110: NORMAL
850  GOSUB 110: VTAB 4: CALL  - 958
860  PRINT : HTAB 2: INVERSE : PRINT "DEC
     ";: HTAB 8: PRINT "HEX";: HTAB 22: PRINT
     "DEC";: HTAB 28: PRINT "HEX": NORMAL
     : PRINT
870  FOR I = 1 TO 14
880  PRINT DOS$(I,2)" ";:DOS =  VAL (DOS$
     (I,2)): GOSUB 310: PRINT HEX$" "DOS$
     (I,1);: HTAB 21: PRINT DOS$((I + 14)
     ,2)" ";:DOS =  VAL (DOS$((I + 14),2)
     ): GOSUB 310: PRINT HEX$" "DOS$((I +
     14),1): NEXT
890  IF FF THEN  FOR I = 1 TO 5: PRINT : NEXT
     : RETURN
900  VTAB 22: PRINT "LIST TABLE TO PRINTE
     R (Y/N) ? ";: GET I$
910  IF I$ = "Y" THEN FF = 1: HTAB 1: CALL
      - 998: CALL  - 958: PRINT B$: INVERSE
     : PRINT " TURN PRINTER ON AND PRESS
     ANY KEY ": PRINT : HTAB 10: PRINT "
     EXPECT A PAUSE ";: GET I$: PRINT : NORMAL
     : PRINT D$;DOS$(20,1);1: PRINT : GOSUB
     810:FF = 0: PRINT D$;DOS$(20,1);0: GOTO
     610
920  IF I$ = "N" THEN 610
930  HTAB 1: GOTO 900
1000  REM  *** CHANGE TABLE ***
1010  HOME :TT$ = "================": GOSUB
     3110
1020 TT$ = "CHANGE COMMANDS": GOSUB 3110
1030 TT$ = "================": GOSUB 3110
1040  VTAB 4: CALL  - 958: VTAB 8: INVERSE
     :TT$ = " READING DOS COMMAND TABLE "
```

```
     : GOSUB 3110: NORMAL
1050  GOSUB 110: VTAB 5: CALL  - 958
1060  FOR I = 1 TO 7
1070  PRINT DOS$(I,1);: HTAB 10: PRINT DO
     S$((I + 7),1);: HTAB 20: PRINT DOS$(
     (I + 14),1);: HTAB 30: PRINT DOS$((I
      + 21),1): NEXT
1080  VTAB 14: CALL  - 958: INPUT "TYPE C
     OMMAND TO BE CHANGED: ";OC$: IF OC$ =
     "" THEN 1180
1090 OC$ =  MID$ (OC$,1, LEN (OC$) - 1) +
      CHR$ ( ASC ( RIGHT$ (OC$,1)) + 128)
     : REM TURN HI BIT ON IN LAST LETTER
     OF    COMMAND
1100  FOR I = 1 TO 28: IF OC$ = DOS$(I,1)
      THEN PT = I: GOTO 1130: REM PT=POIN
     TER TO      POSITION OF COMMAND IN A
     RRAY
1110  IF I = 28 THEN  PRINT B$: VTAB 16: INVERSE
     : PRINT " NOT A VALID CURRENT COMMAN
     D ": NORMAL : FOR J = 1 TO 3000: NEXT
     : GOTO 1080
1120  NEXT I
1130  VTAB 16: CALL  - 958: INPUT "TYPE N
     EW COMMAND: ";NC$: IF NC$ = "" THEN
     1130
1140 NC$ =  MID$ (NC$,1, LEN (NC$) - 1) +
      CHR$ ( ASC ( RIGHT$ (NC$,1)) + 128)
     : REM TURN HI BIT ON IN LAST LETTER
     OF    COMMAND
1150  PRINT B$: VTAB 18: HTAB 3: PRINT "C
     ONFIRM (Y/N) ? ";: GET I$: PRINT I$
1160  IF I$ = "Y" THEN  VTAB 20: INVERSE
     : PRINT " WRITING COMMAND TABLE ": GOSUB
     410: VTAB 18: HTAB 1: CALL  - 958: PRINT
     " CHANGE COMPLETED ": NORMAL : GOTO
     1220
1170  IF I$ < > "N" THEN  VTAB 18: CALL
      - 958: GOTO 1150
```

```
1180  VTAB 18: CALL  - 958: PRINT : PRINT
     "RETURN TO MENU OR TRY AGAIN (M/A) ?
      ";: GET I$: PRINT I$
1190  IF I$ = "A" THEN  GOTO 1080
1200  IF I$ = "M" THEN 610
1210  GOTO 1180
1220  VTAB 20: CALL  - 958: PRINT "ANOTHE
     R CHANGE (Y/N) ? ";: GET I$: PRINT I
     $: IF I$ = "Y" THEN 1040
1230  IF I$ = "N" THEN 610
1240  GOTO 1220
2000  REM  *** INITIALIZE ***
2010  DIM DOS$(30,2)
2020 D$ =  CHR$ (4):B$ =  CHR$ (7):SS$ =
     "                     ": REM 21 SPAC
     ES
2030 H$ = "0123456789ABCDEF"
2040  DEF  FN MOD(X) = X -  INT (X / 256)
      * 256: REM SIMULATE MOD FUNCTION
2050 START = 43140: REM START OF TABLE
2060  RETURN
3110 WIDTH = 20 - ( LEN (TT$) / 2): IF WI
     DTH <  = 0 THEN  PRINT TT$: RETURN
3120  HTAB WIDTH: PRINT TT$: RETURN
```

程式 5－4　DOS命令修改的程式

DOS 命令名稱的 ASCII 碼表是存於＄A884 至＄A908（ 43140 至 43272 ）之間，總共有 28 個命令，每個名稱只有最後字母是以負ASCII 碼來表示的， DOS 一檢查到負ASCII 碼便知道是命令名稱的結束，而整體命令表則以 0 做爲結尾。本程式便是利用這種特性來處理 DOS 的命令名稱表，譬如110 行設定 TM＝START＝43140 ， 然後120 及130 行便由此開始來

找出命令表的結尾並設定成 FIN ，而150 至180 行的迴圈則 PEEK由START 至 FIN 的每一位址來累積成字串，若遇到負 ASCII 碼則表示字串的結束，以DOS$（（I＋1），2）＝ STR$（DOS＋1）來記錄位址後再進行下一名稱字串，而 DOS$ 則記錄整串名稱表的內容。

修改命令名稱時，由1100 行來找出將被更換的命令名稱所在，而以變數PT 來記錄其號碼（1～28），1140 行將新名稱最後字元改爲負ASCII 碼，然後 GOSUB 410 去做命令名稱更換的工作。由於命令名稱長度的改變，因此以 410 至460 行來造出新名稱表字串DOS$，由480 行 POKE 回名稱表位址，最後以 500行記下結束記號 0 。

執行本程式時，請注意自己設定的名稱切勿與BASIC 指令有所重合，譬如RENAME 若簡寫成 RE 會與REM重合、LOAD 若簡寫成L 會與 LIST 重合，這是因爲所有立即命令均要經過 DOS 的KBDINCPT 攔截，一旦 DOS 看到LIST 時會將它當成L（ OAD ）IST （因爲新命令表中有L這個名稱），結果當然是FILE NOT FOUND 了。

5－6 創造DOS的命令：TYPE文字檔

使用APPLE DOS 的人都會感覺到，要將一個文字檔的內容印出到螢幕或印字機是相當麻煩的事情，因爲至少要寫一個程式將它讀出來才行。如果能夠像CP／M 的TYPE、PASCAL的T（RANSFER或UNIX的cat ，只要鍵入命令就能印出檔案內容，那眞是很方便呢！底下的這個程式可以令您只鍵入' TYPE

檔名，就可將各種類型的檔案內容印出來：

```
*****************************
**    FILE ---> DUMP PROG   **
*****************************
OPEN       EQU $A2A8
CLOSE      EQU $A2FC
READ       EQU $A68C
ZEROES     EQU $AA62
* COMMAND LINE
RETCODE    EQU $B5C5
KBD        EQU $C000
KBDSTRB    EQU $C010
CROUT      EQU $FD8E
COUT       EQU $FDED
           ORG $BC56
TYPE       JSR OPEN
           JSR CROUT
NEXT       JSR READ
           ORA #$80
           JSR COUT
ZERRTS     LDA KBD
           BPL RTS
           BIT KBDSTRB
           CMP #$83
           BEQ DONE
           CMP #$93
           BNE RTS
WTLOOP     LDA KBD
           BPL WTLOOP
           BIT KBDSTRB
           CMP #$83
           BEQ DONE
RTS        LDA RETCODE
           CMP #$05
           BNE NEXT
DONE       JSR CROUT
```

```
37          JMP CLOSE
38          ORG $9D1E
39          ADR TYPE-1
40          ORG $A884
41          DCI 'TYPE'
42          END
```

程式5－5　TYPE TEXT_FILE

```
*BC56.BC8C

BC56- 20 A8
BC58- A2 20 8E FD 20 8C A6 09
BC60- 80 20 ED FD AD 00 C0 10
BC68- 17 2C 10 C0 C9 83 F0 17
BC70- C9 93 D0 0C AD 00 C0 10
BC78- FB 2C 10 C0 C9 83 F0 07
BC80- AD C5 B5 C9 05 D0 D5 20
BC88- 8E FD 4C FC A2
*9D1E.9D1F

9D1E- 55 BC
*A884.A887

A884- 54 59 50 C5
```

程式5－5的機械碼

本程式主要利用＄A2A8的OPEN ANYTHING來開啓任何型式的檔案，所以無論A、B、I、T、L、R 檔均可以TYPE出來，不過由於文字檔以外的檔案內容不全是文數字（譬如代碼），因而會有奇怪的結果出現。

程式的主廻圈是由18行至35行構成，18行JSR READ由檔案中讀了一個字元，將最高位元設為1之後，便以COUT印了出來，由於最高位元為1，因而TYPE出來的均為正常字形，較方便閱讀。21至32行是附加的CTRL-S及CTRL-C功能，TYPE時按下CTRL-S可以使螢幕的列印動作暫停，而按下CTRL-C則結束列印過程。一旦READ($A2A8)讀完檔案，其返回碼會設定成$05，34行便是用來檢知是否檔案已結束，若是，則JMP CLOSE關閉檔案。

由上一節我們知道DOS的命令名稱由$A884開始存放，我們的TYPE恰好將INIT覆蓋住，因此INIT便失效了，但TYPE却變成正式的DOS命令，可以很方便的直接下達。至於$9D1E是原來INIT程式的進入點，我們將它換成TYPE，39行使用位址減一，是因為APPLE是以堆疊及RTS來做跳躍動作的緣故*。

參考文獻：

Ref 1. APPLE II DOS 使用手册

Ref 2. APPLE II 微電腦徹底研究(二)，施威銘著，波前 1982

Ref 3 APPLE II 機械語言技術，施威銘著，波前 1984

Ref 4. Call APPLE all about DOS

---

*以堆疊及RTS來做ON S GOTO 的動作，請參閱APPLE II機械語言技術，施威銘著，波前 1984

本章提要

對 DOS 稍有瞭解的人都知道RWTS是 APPLE 實際的 I／O程式，RWTS內部如何動作呢？本章將RWTS完全解剖，令您完全揭開RWTS的神秘細節：

①6／2 編碼法

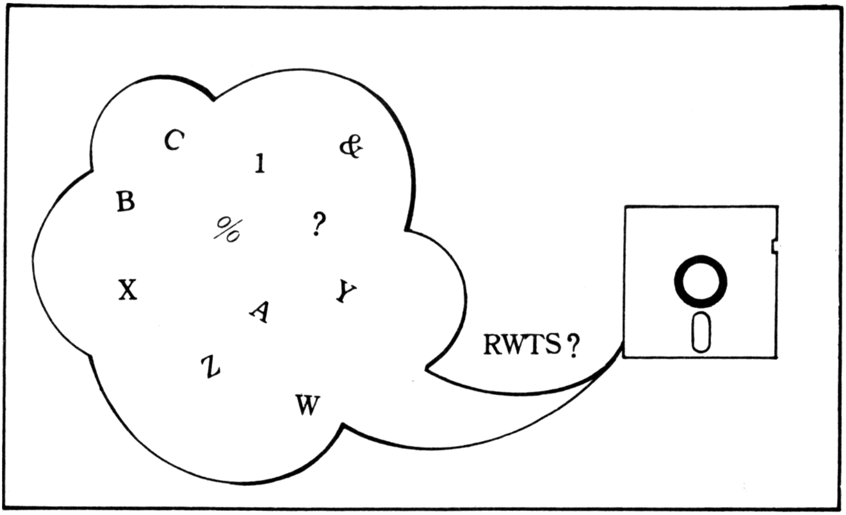
②PRE－NIBL、POST－NIBL

③READ、WRIT

④查表轉換、查表還原

⑤………………

6 RWTS徹底研究



6－0 6/2編碼法

在第 0 章我們分析了 DOS 的 READADDR 及 SEEKABS 這兩個最基層的副程式，它們的用途是讀取磁頭的位置座標（V，T，S），以及移動磁頭至指定的位置。一旦磁頭到達了目的位置之後，接下來的便是在該位置讀寫資料了，由於資料欄所要存放的位元組（ 256 byte ）遠多於位址欄，若使用第 0 章提到的奇偶編碼法勢必浪費空間，因此採用了一種稱為6／2編碼的方法。使用這種方法，256 個資料位元組編碼後變成 342 個位元組

，比奇偶編碼所造成的 512 個位元組節省了許多空間。

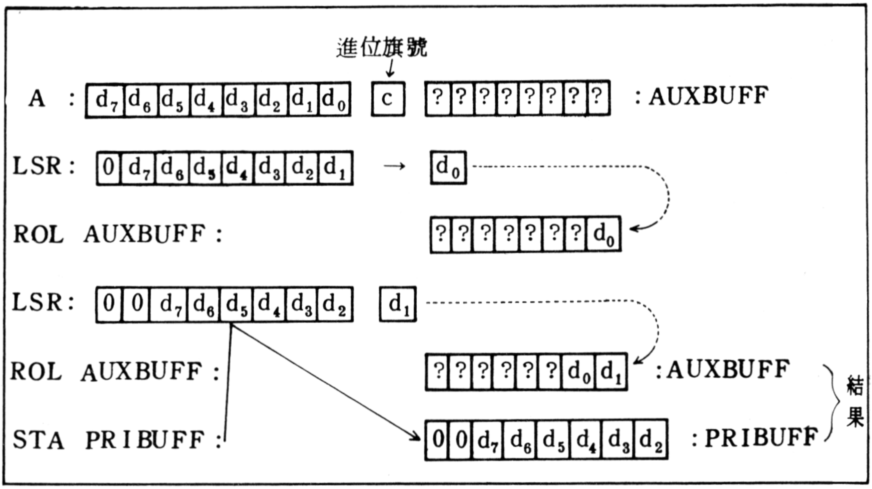


圖 6－0　將 A 的內含值 6／2 編碼後存入 PRIBUFF 及 AUXBUFF 的情形

上圖便是 6／2 編碼的過程，每個位元組分成 6 與 2 個位元，前（高）6 個位元存於主編碼區（PRIBUFF），其餘兩個位元則存入輔助編碼區（AUXBUFF），因此輔助編碼區裏每個位元組可存 3 個正常位元組的後（低）2 位元。

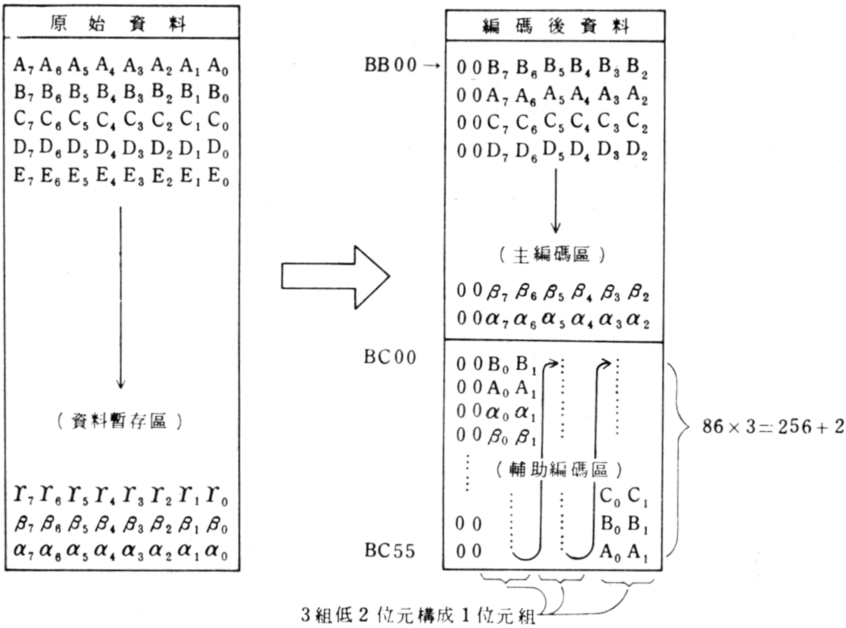


圖 6 − 1　編碼區的存放情形

將資料位元組編碼之後，還不能用來寫入磁片，因爲圖 6 − 1 編碼後的最高兩位元爲 0 ，但 APPLE 硬體的限制是：

1. 最高位元必須是 1
2. 最多連續兩個 0 ，並且只能有一對
3. 至少要有一對連續的 1

滿足這些條件的位元組是 \$96 （10010110）、 \$97 、 \$9A ………，譬如 \$95 = 10010101 因爲沒有一對連續的 1 ，因此不符條件。這樣的位元組最小爲 \$96 最大爲 \$FF 共 66 個，我

們將其中的 $AA 及 $D5 保留爲特殊的指示位元組，其餘的 64 個位元組恰可與編碼後由 6 個位元所組成共 $2^6=64$ 個位元組做一對一對應（編碼後的位元組是 $0 ～ $3F，因最高 2 位元爲 0 ），這個對應表列出如下*：

6／2編碼對應表

00 ⇨ 96	10 ⇨ B4	20 ⇨ D6	30 ⇨ ED
01 ⇨ 97	11 ⇨ B5	21 ⇨ D7	31 ⇨ EE
02 ⇨ 9A	12 ⇨ B6	22 ⇨ D9	32 ⇨ EF
03 ⇨ 9B	13 ⇨ B7	23 ⇨ DA	33 ⇨ F2
04 ⇨ 9D	14 ⇨ B9	24 ⇨ DB	34 ⇨ F3
05 ⇨ 9E	15 ⇨ BA	25 ⇨ DC	35 ⇨ F4
06 ⇨ 9F	16 ⇨ BB	26 ⇨ DD	36 ⇨ F5
07 ⇨ A6	17 ⇨ BC	27 ⇨ DE	37 ⇨ F6
08 ⇨ A7	18 ⇨ BD	28 ⇨ DF	38 ⇨ F7
09 ⇨ AB	19 ⇨ BE	29 ⇨ E5	39 ⇨ F9
0A ⇨ AC	1A ⇨ BF	2A ⇨ E6	3A ⇨ FA
0B ⇨ AD	1B ⇨ CB	2B ⇨ E7	3B ⇨ FB
0C ⇨ AE	1C ⇨ CD	2C ⇨ E9	3C ⇨ FC
0D ⇨ AF	1D ⇨ CE	2D ⇨ EA	3D ⇨ FD
0E ⇨ B2	1E ⇨ CF	2E ⇨ EB	3E ⇨ FE
0F ⇨ B3	1F ⇨ D3	2F ⇨ EC	3F ⇨ FF

AA
D5 保留位元組

圖 6－2 將 6／2 編碼後的位元組再轉換成符合硬體要求的碼

上圖中， 00 編碼後變成 96 ，這正是程式 0 － 0 讀到一大堆 96 （圖 0 － 11 ）的原因！

＊此即 GCR 編碼表，參考附錄 A

6－1 編碼程式：PRE-NIBBLE

圖6－1的編碼及存放方式看似很複雜，到底程式上是怎樣來安排的呢？底下是 DOS RWTS 的編碼程式（PRE_NIBBLE）：

PRE_NIBBLE

```
*------------------------------------------
* ZERO PAGE VARIABLES
*------------------------------------------
BUFINDEX EPZ $26            ;暫存區索引
ADRCHECK EPZ $27
SLTINDEX EPZ $27
SCRATCH1 EPZ $26
SCRATCH2 EPZ $27
SCRATCH3 EPZ $2A
SCRATCH4 EPZ $2B
CHKSUM   EPZ $2C
SECTOR   EPZ $2D
TRACK    EPZ $2E
VOLUME   EPZ $2F
BUFFPTR  EPZ $3E            ;暫存區指位器
MONTIME  EPZ $46
*------------------------------------------
* DOS VARIABLES
*------------------------------------------
CURTRK   EQU $0478
ALTINDEX EQU $0678
*------------------------------------------
* HARDWARE ADDRESSES
*------------------------------------------
```

本程式為說明方便共分成3則，由程式6－0至6－2分別列出。

```
PHASEOFF EQU $C080
Q6L      EQU $C08C
Q6H      EQU $C08D
Q7L      EQU $C08E
Q7H      EQU $C08F
*
*  Q6    Q7           控制線的使用法
*  ----  ----  -----------------------------
*  LOW   LOW   Read (disk to shift register)
*  LOW   HIGH  Write (shift register to disk)
*  HIGH  HIGH  Load shift register from data bus
*  HIGH  LOW   sense write protect
*
**********************************
* NOW DIFINE PRIMARY BUFFER AND  *
* AUXILARY BUFFER ......         *
**********************************
              ORG $BB00
PRIBUFF       DFS $100      ；主暫存區由＄BB00～＄BBFF
*
AUXBUFF       DFS $56       ；輔助暫存區＄BC00～＄BC55
**********************************
* THE PRE_NIBBLE ROUTINE CONVERTS *
* 256 BYTES IN THE SECTOR BUFFER  *
* TO 342 6-BIT NIBBLES IN PRIMBUFF*
* AND AUXBUFF.                    *
**********************************
              ORG $B800
PRENIBL       LDX #$00
              LDY #$02      ；迴圈3次，一次編碼86位元組
NEXTBYTE      DEY           ；減258次＝0
              LDA (BUFFPTR),Y   ；由IOB所設定的資料
              LSR               暫存區取資料
              ROL AUXBUFF,X
              LSR               執行圖6－0的動作
              ROL AUXBUFF,X     （寫入編碼值）
              STA PRIBUFF,Y
              INX
```

```
64            CPX #86            ；已編碼了86位元組？
65            BCC NEXTBYTE
66            LDX #$00
67            TYA
68            BNE NEXTBYTE
69            LDX #85            ；AUXBUF有86位元組
70 TRUNCATE   LDA AUXBUFF,X
71            AND #$3F           ；設b6b7為0
72            STA AUXBUFF,X
73            DEX
74            BPL TRUNCATE
75            RTS
```

程式6－0　6／2編碼程式

這就是RWTS的編碼程式，正如圖6－0所示的，資料位元組的低2位元存入輔助編碼區（$BC00或$BC55），而其餘6個位元存入主暫存區（$BB00至$BBFF），將程式57行至62行與圖6－0比照就可以瞭解。本程式要特別注意的是NEXTBYTE迴圈，進入NEXTBYTE前，Y的值為2、X的值為0，每次Y減一而X加一，一直到X等於86為止。因此，迴圈完成一次共處理了86個位元組，在第3次時Y的值恰好為0（86＊3－2＝256→0），這時編碼工作就結束了。這樣的做法是為了縮短程式，但卻使編碼區的構造趨於複雜化，其中，主編碼區與資料暫存區的資料次序是相同的，這是因為二者均以Y暫存器做為索引（57及62行），因此雖然填了前2位元組後（Y由2開始遞減）馬上跳至第255位元組，但每個位元組的相互次序是不變的。

至於輔助編碼區，因為是以X暫存器做索引並且每次只填2

位元，所以第一次塡入的位置是 \$BC00 的 B_0B_1 位元（第 4－5 位元，見圖 6－1），然後是 \$BC01 的 A_0A_1 ，依此類推，共在 86 個位元組上塡入了 256 個資料位元組的低 2 位元，由於迴圈是 3 × 86 ＝ 258 次，所以最初 2 個位元組資料是重複一次。整個輔助編碼區的存放次序是如圖 6－1 的箭頭所示，這是 59 及 61 行的 ROL 所致。

通常這些已編碼的位元組，實際上只隱含了眞正資料位元組的 6 或 2 個位元而已，所以稱爲 Nibble ，而這個編碼程式便叫做 Pre - nibble。一些 COPY 程式便是因其是在 nibble （未解碼）階層工作，故稱爲 Nibble COPY 。

6－2 RWTS的區段寫出:BLOCK WRITE

既然已將資料編碼完畢，接下來的工作就是將主暫存區及輔助暫存區寫入磁片，這個工作是由 \$B82A 至 B8C1 的 WRITE 程式來執行的：

WRITE

```
***********************************
*THE WRITE SUBROUTINE WRITES A    *
* BLOCK TO DISK CONSISTING OF :   *
*A)THE PROLOGUE
*B)A SERIAL EQUIVALENT OF THE DATA*
*  NIBBLE BUFFERS
*C)THE CHECKSUM                   *
*D)THE EPILOGUE (DE AA EB)        *
***********************************
```

```
WRITE      SEC  ($B82A); C＝1，若 err 則傳回呼叫程式
           STX  SLTINDEX ; 設定 slot
           STX  ALTINDEX ; 存入另一個 slot 變數
           LDA  Q6H,X    }
           LDA  Q7L,X    }; 是否貼上保護片？
           BMI  RETURN1  ; 若是，C＝1，至Return1
           LDA  AUXBUFF  ; 取第一個資料
           STA  SCRATCH1 ; 暫存
           LDA  #$FF
           STA  Q7H,X; 準備寫出
           ORA  Q6L,X; 第一個FF  ┐
           PHA                   │
           PLA                   │ 延時
           NOP                   │
           LDY  #$04  ; 寫4個FF  ┘
WSYNC      PHA (延時)  ; 3 cycles     ┐
           PLA (延時)  ; 4 cycles     │
空出2個    JSR  WRITE2; 6 cycles，寫一個FF  │
0位元      DEY         ; 2 cycles     │
           BNE  WSYNC  ; 3 cycles     ┘
32 cycles  LDA  #$D5   ; 2 cycles     ┐
           JSR  WRITE1; 6 cycles，寫D5  ┘
           LDA  #$AA
           JSR  WRITE1          ; 寫AA
           LDA  #$AD
           JSR  WRITE1          ; 寫AD
           TYA                  ; A←0
           LDY  #86             ; 寫86個編碼區的值
           BNE  EORNIBL         ; JMP EORNIBL
WRITEAUX   LDA  AUXBUFF,Y                      ┐
EORNIBL    EOR  AUXBUFF-1,Y; 由$BC55開始造checksum │
           TAX             ; 以EOR後A的值爲索引  │
           LDA  NIBLTBL,X  ; 取得對應值          │
32 cycles  LDX  SLTINDEX   ; 然後                │
           STA  Q6H,X      ; 由主機送出          │
           LDA  Q6L,X      ; 磁頭動作            │
           DEY                                   │
           BNE  WRITEAUX        ; 86次           ┘
```

```
123                LDA  SCRATCH1     ；取 AUXBUF 的最前頭
124                NOP               ；位元組（也是最後）
125 WRITEPRI       EOR  PRIBUFF,Y    ；由 $BB00 開始造 checksum
126                TAX
127                LDA  NIBLTBL,X
128                LDX  ALTINDEX
129                STA  Q6H,X        ；送出資料至週邊卡
130                LDA  Q6L,X        ；令磁頭動作寫至磁片
131                LDA  PRIBUFF,Y
132                INY
133                BNE  WRITEPRI     ；256次？
134                TAX               ⎫
135                LDA  NIBLTBL,X    ⎬ 寫 checksum
136                LDX  SLTINDEX     ⎪
137                JSR  WRITE3       ⎭
138                LDA  #$DE         ⎫
139                JSR  WRITE1       ⎪
140                LDA  #$AA         ⎬ 寫結束區
141                JSR  WRITE1       ⎪
142                LDA  #$EB         ⎪
143                JSR  WRITE1       ⎪
144                LDA  #$FF         ⎪
145                JSR  WRITE1       ⎭
146                LDA  Q7L,X        ；關掉 write 狀態
147 RETURN1        LDA  Q6L,X        ；還原至 read 狀態
148                RTS
149 ***********************************
150 * WRITE A BYTE & DELAY  ....      *
151 ***********************************
152 WRITE1         CLC           ； 2 cycles
153 WRITE2         PHA           ； 3 cycles
154                PLA           ； 4 cycles
155 WRITE3         STA  Q6H,X ；送出資料（Q7H已在94行設定）
156（本副程式共）ORA  Q6L,X；磁頭動作， 4 cycles  ↲， 5 cycles
157（ 24 cycles ）RTS           ； 6 cycles
```

程式 6－1

為了瞭解上述程式，首先我們要知道磁頭的讀寫控制法。由圖0－10我們知道磁碟控制卡上的16個位址共分成4部份，C0X0至C0X7用來移動磁頭，C0X8及C0X9用來控制磁片旋轉，C0XA及C0XB則選擇磁碟機（1或2號），至於C0XC至C0XF就是此處要討論的磁頭控制位址：

Q7H（C0XF）	：指示控制卡準備並等候磁頭的寫出動作。
Q7L（C0XE）	：指示控制卡準備並等候磁頭的讀入動作。
Q6H（C0XD）	：將資料送往控制卡暫存器以便寫出。
Q6L（C0XC）	：令磁頭做讀寫動作。

＊這些位址是用來控制DISK介面板上9334IC（LS259）的Q6及Q7之輸出電位，因而以Q6H、L及Q7H、L稱之。

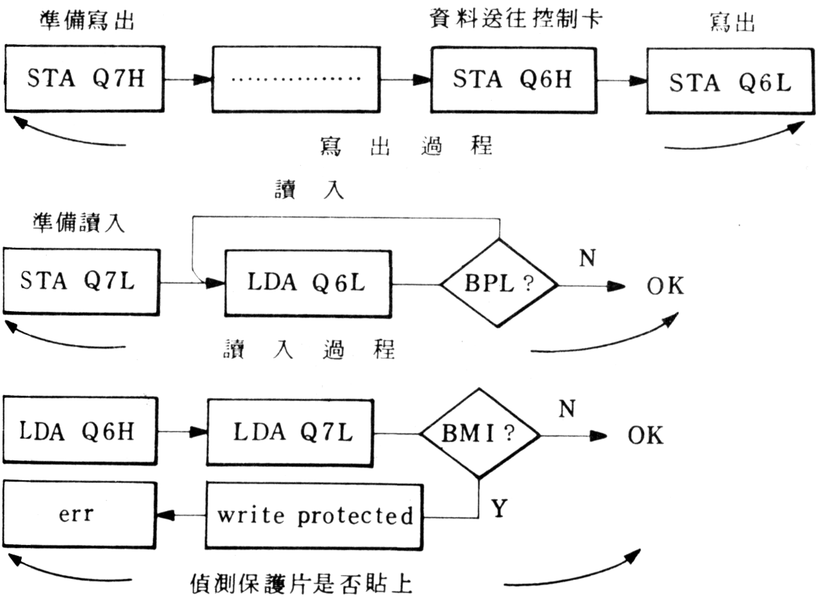
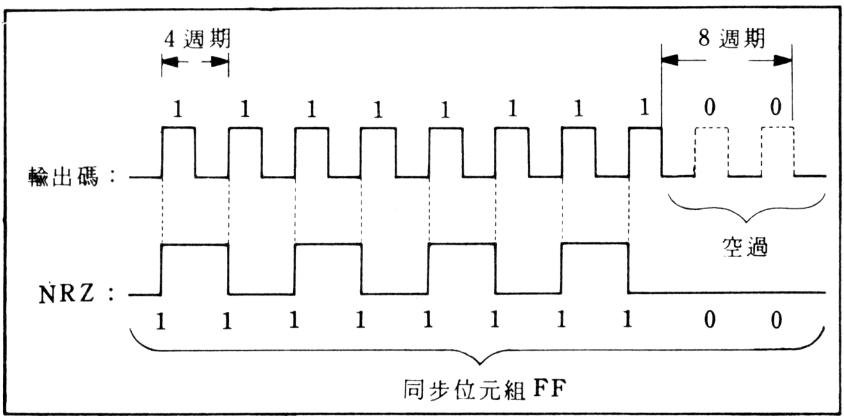


圖6－3

由上面的圖示，我們就可以來看 WRITE 程式的動作過程了。為了安全起見， DOS 啓動後，磁頭是設定於READ的狀態，並且由程式的 146 、 147 行看到，每次 WRITE 之後也要還原至 READ。程式首先由 88 、 89 行來檢視磁片是否貼上保護貼紙，若已貼上，則不還原 85 行的 SEC （錯誤旗號）而至 RETURN1 。要寫出資料之前必須先寫出同步區，這是由 93 至 104 行來完成的，這段程式共寫出 5 個同步位元組 FF ，仔細計算其時間間隔（參考程式註解）可以發現共有 40 cycles 的間距，由於 APPLE 是以時序控制器*來寫資料的，因而寫一個位元要用掉 4 個時脈週期，8 個位元共 32 個週期，其餘還有（ 40 − 32 ）／ 4 ＝ 2 位元空隔就是 0 − 3 節所提到的同步 0 位元：



至於接下來的 D5 、 AA 、 AD 前導區，則每位元組（ 8 位元）均以 32 週期的時間來寫下其 8 個位元的資料，譬如 105 、

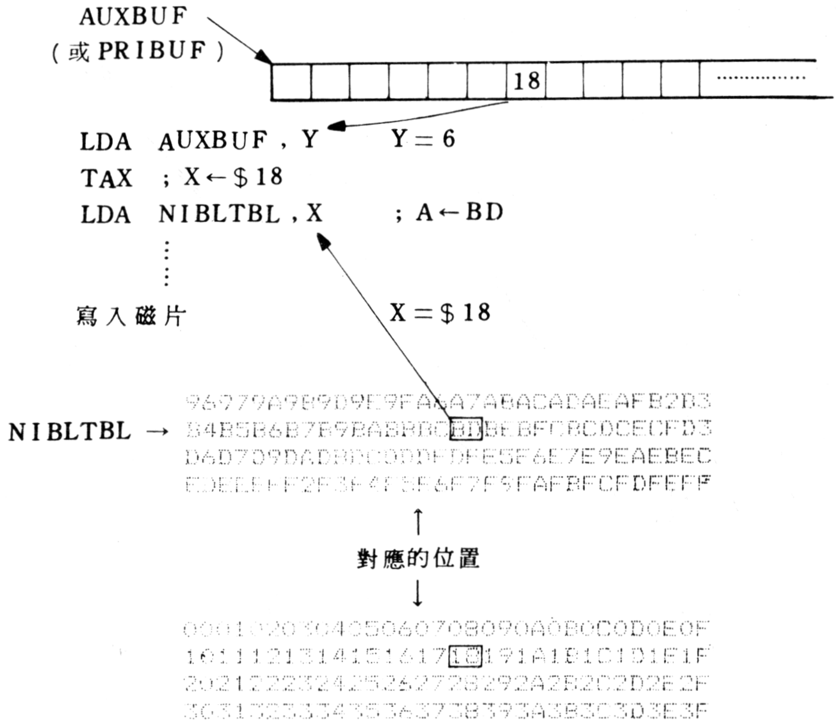
---

*附錄 A ，GCR的基本動作是 NRZ ，NRZ 對於沒有訊號的情況是當成 0 來處理的。

106 行便是。寫完前導區後，接著就由 114 行的 WRITEAUX 及 125 行的 WRITEPRI 程式來將輔助編碼區及主編碼區寫入磁片，由這些程式您可以發現，RWTS爲何是區段 I／O而不能做各別字元的 I／O了。

資料轉換法

雖然資料暫存區的內含值已經由 PRE _ NIBBLE 編碼至主、輔編碼區了，可是還要經由圖 6 – 2 轉換成可以寫入磁片的碼才能存至磁片，爲了節省程式空間及速度， APPLE 是將轉換及寫出的動作一併完成的。到底 APPLE 如何來轉換這些資料呢？其方法是很巧妙的，由圖6 – 2 我們可以看出，左邊的碼是由0至 3F 依次排列，要找 08 的轉換值，只要由右邊的碼算到第 8 個（由 0 算起）位置上便找到了。這是說，如果我們將右邊的碼列成一個表格（即 243 至 246 行的NIBLTBL ，見下一程式），要找 n 這個碼的轉換值，則只要以 n 做索引來取表，便可以了：



6 － 4　查表轉換法（Table lookup transform）

上圖的例子是由編碼區內取出一個資料碼 $18 存入 X，然後以 LDA NIBLTBL ，X的方式來找出NIBLTBL第 $18 位置上的碼 $BD ，對照圖 6 － 2，您可以看出 $18 所對應的碼眞的是 $BD 無誤。

CHECHSUM的造法

除了以查表轉換法來取得NIBLTBL上的合法位元組外，我們還要造出資料磁區的偵錯區位元組（checksum）（見圖0－4）。可是每次寫一位元組一定要不多不少的間隔 32 時脈週期，以使磁頭可以完全的寫完一位元組的 8 個位元，對於像 114 行至 122 行這樣的程式，實在是無法再抽出空檔來做 checksum 處理了。因此，APPLE 的處理方法是首先將 0 與編碼區最前頭的位元組做 EOR，然後將結果再依次與下一位元組 EOR 下去，當處理完全部資料時，最後的結果就是 checksum 了（134 至 137 行）。在這個過程當中，其特異處是 APPLE 寫入磁片的竟然是每次 EOR 的值而不是資料本身！

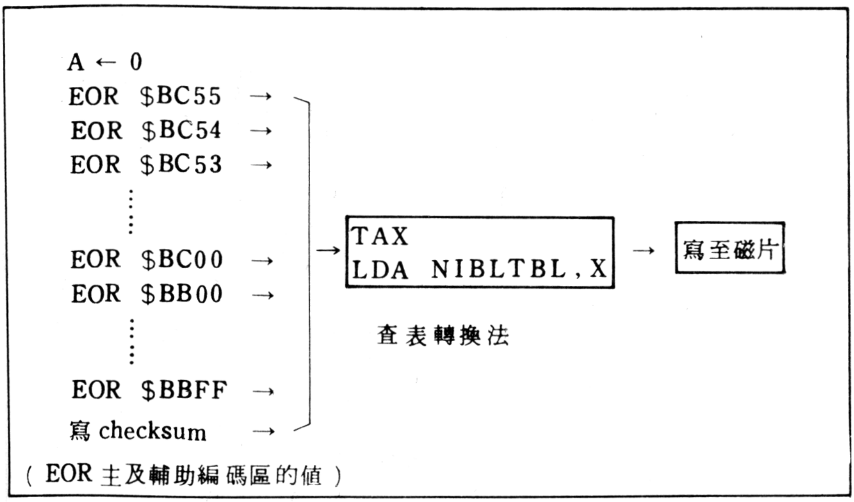


圖6－5　以EOR的結果寫入磁片

由於 EOR 指令的特殊，兩次的 EOR 會將原來的值重現：

A	:	11011100
B	:	01110011
A∀B	:	10101111
(A∀B)∀B	:	11011100=A

＊參考第5冊 2－3節

因此，雖然寫入磁片的是 EOR 的值，只要讀出後再 EOR 一次，就可以取回原值了：

由磁片讀取 → 查表還原法 →

```
A← 0
EOR  $BC55 → $BC55
EOR  $BC55 → $BC54
        ⋮
        ⋮
EOR  $BC01 → $BC00
EOR  $BC00 → $BB00
EOR  $BB00 → $BB01
        ⋮
        ⋮
EOR  $BBFE → $BBFF
EOR  $BBFF → 必須是 0
```

＊圖6－5中，$BC54是與上一位元組 $BC55 EOR，因此還原時，要與 $BC55 EOR 才能還原$BC54

圖6－6　再次EOR來還原資料

所以程式6－1就可以清楚瞭解了，114至137行的動作原理參照圖6－4、5、6是不難瞭解的。由138至145是寫結束區的訊號，請注意144行多寫了一個FF位元組，因此圖0－11的資料欄結束區最後位元組EB才不受破壞。

6－3 RWTS的區段讀入:BLOCK READ

接著，我們來看寫出的資料如何讀入。讀入動作是與前節的寫出過程相反的，因此瞭解了上一節的原理之後，一切就比較簡單了。

READ與POSTNIBL

```
158 **********************************
159 * THE POST NIBBLE SUBROUTINE WILL*
160 * CONVERT 342 NIBBLES IN PRIMBUFF*
161 * AND AUXBUFF TO 256 BYTES IN THE*
162 * SECTOR BUFFER.                 *
163 **********************************
164 POSTNIBL  LDY #$00 ($B8C2)
165 CNVBLK    LDX #$56
166 CNVCHK    DEX
167           BMI CNVBLK
168           LDA PRIBUFF,Y
169           LSR AUXBUFF,X
170           ROL
171           LSR AUXBUFF,X
172           ROL
173           STA (BUFFPTR),Y
174           INY
175           CPY SCRATCH1
176           BNE CNVCHK
177           RTS
```

（164～177行：與PRENIBL的動作相反）

```
************************************
* THE READ SUBROUTINE READS A DATA*
* BLOCK FROM THE DISK AND STORES   *
* THE DATA NIBBLES INTO PRIBUFF    *
* AND AUXBUFF.                     *
************************************
READ      LDY #32                 ($B8DC)
D5LOOP    DEY
          BEQ RETURN2             ;嘗試32次
READD5    LDA Q6L,X               ;磁頭動作
          BPL READD5
TESTD5    EOR #$D5                ;D5?
          BNE D5LOOP
          NOP                     ;延時
READAA    LDA Q6L,X
          BPL READAA
          CMP #$AA                ;AA?
          BNE TESTD5              ;否，重新開始
          LDY #86                 ;延時，Y=86
READAD    LDA Q6L,X
          BPL READAD
          CMP #$AD                ;AD?
          BNE TESTD5              ;否，重新開始
          LDA #$00                ;延時，A=0
RAUXBUFF  DEY                     ;開始讀回輔助編碼區
          STY BUFINDEX
RSERIAL1  LDY Q6L,X               ;讀資料到Y
          BPL RSERIAL1            ;Y的值一定大於$96
          EOR BYTETBL-$96,Y;將A與BYTETBL的值EOR
          LDY BUFINDEX            ;取索引
          STA AUXBUFF,Y           ;由$BC55往前填入
          BNE RAUXBUFF
RPRIBUFF  STY BUFINDEX            ┐
RSERIAL2  LDY Q6L,X               │
          BPL RSERIAL2            │
          EOR BYTETBL-$96,Y       │ 讀回主編碼區
          LDY BUFINDEX            │
          STA PRIBUFF,Y           │
          INY                     │
          BNE RPRIBUFF            ┘
```

```
TESTCHK   LDY Q6L,X            ; 檢查 checksum
          BPL TESTCHK
          CMP BYTETBL-$96,Y
          BNE RETURN2          ; error
TESTDE    LDA Q6L,X
          BPL TESTDE
          CMP #$DE             ; DE ?
          BNE RETURN2
          NOP
TESTAA    LDA Q6L,X
          BPL TESTAA
          CMP #$AA             ; AA ?
          BEQ RETURN3          ; OK !
RETURN2   SEC                  ; on err set c
          RTS
          ORG $B99E
RETURN3   CLC                  ; OK , clear c
          RTS
          ORG $BA29
***********************************
*THIS TABLE IS USED BY THE WRITE  *
*ROUTINE TO CHANGE NIBBLE VALUES  *
*FROM $00 TO $3F INTO ONE OF 64   *
*LEGAL BYTES.                     *
***********************************
NIBLTBL   HEX 96979A9B9D9E9FA6A7ABACADAEAFB2B3
          HEX B4B5B6B7B9BABBBCBDBEBFCBCDCECFD3
          HEX D6D7D9DADBDCDDDEDFE5E6E7E9EAEBEC
          HEX EDEEEFF2F3F4F5F6F7F9FAFBFCFDFEFF
NOTUSED   DFS $2D              ( $BA69～$BA95 )
***********************************
* TRANSLATION TABLE : CONVERT $96 *
* - $FF INTO ONE OF THE ORINGINAL *
* 64 DATA NIBBLES ($00 - $3F)     *
***********************************
BYTETBL   HEX 00019899020309C040506A0A1A2A3A4A5
($BA96)   HEX 0708A8A9AA090A0B0C0DB0B10E0F1011
          HEX 1213B81415161718191AC0C1C2C3C4C5
```

不會用到的碼補空缺用 (refers to the boxed bytes 9899, 9C, A0A1A2A3A4A5)

```
256              HEX C6C7C8C9CA1BCC1C1D1ED0D1D21FD4D5
257              HEX 2021D8222324252627 28E0E1E2E3E429
258              HEX 2A2BE82C2D2E2F303132F0F133343536
259              HEX 3738F8393A3B3C3D3E3F
260              END
```

程式 6 － 2

本程式銜接著程式 6 － 1 ，由 164 至 177 行是與 PRENIBL 相反的 POSTNIBL ，茲不重述。184 行開始是 READ 程式，由於 WRITE 結束時（ 146 、 7 行）已將磁頭狀態設爲 READ，因此只要觸動 Q6L 便可以讀入訊號了。185 行至 200 行及 222 行至 230 行是用以檢查前導區與結束區，結束區與程式 0 － 2 的 READADDR 一樣仍然只檢查 DE、AA 兩個位元組而已。

202 行至 209 行開始讀取資料至輔助編碼區，其過程正如圖 6 － 6 所示一般，但是請注意此處資料是讀至 Y 暫存器的（ 218 行），其目的是要利用 Y 做爲查表索引，也就是與圖 6 － 4 類似的查表法，可是由磁頭讀至 Y 的是大於 $96 的數值，因此查表時要先減 $96 ，但是爲了保持正確的 32 週期讀取的動作，因此查表時先倒退 $96 個位址（ 213 行的 EOR BYTETBL － $96 , Y ），如此可省下做減法的時間。

除了倒退 $96 個位元組外，由圖 6 － 2 的轉換表可以看出雖然左邊的數值是由 $00 至 $3F 連續排列，但右邊由 $96 至 $FF 的數值是不連續的，因此，我們不能像圖 6 － 4 那樣直接以對應位置來查表。因爲由圖 6 － 4 可以看出， $96 若倒退 $96 個位元組來查表（ $96－$96 ）恰好可以查到 $00 這個

碼，但是$9A－$96＝$04 查到的卻是 $04 而不是 $02 ，結果與圖 6 － 4 不符。

為了解決這個問題，APPLE 重新編造了一個稱為BYTETBL的表格，在 $01 與 $02 間插入兩個無用的碼，如此$9A－$96 查表後便取到 $02 了，其他空缺處也依此原則來填補，由於這些空缺是某些數值所造成的（譬如 $97 與 $9A 間的 $98、$99 ），因此就用這些數來填充，這是為了方便瞭解，其實任何數值均可，因為它們根本不用到。

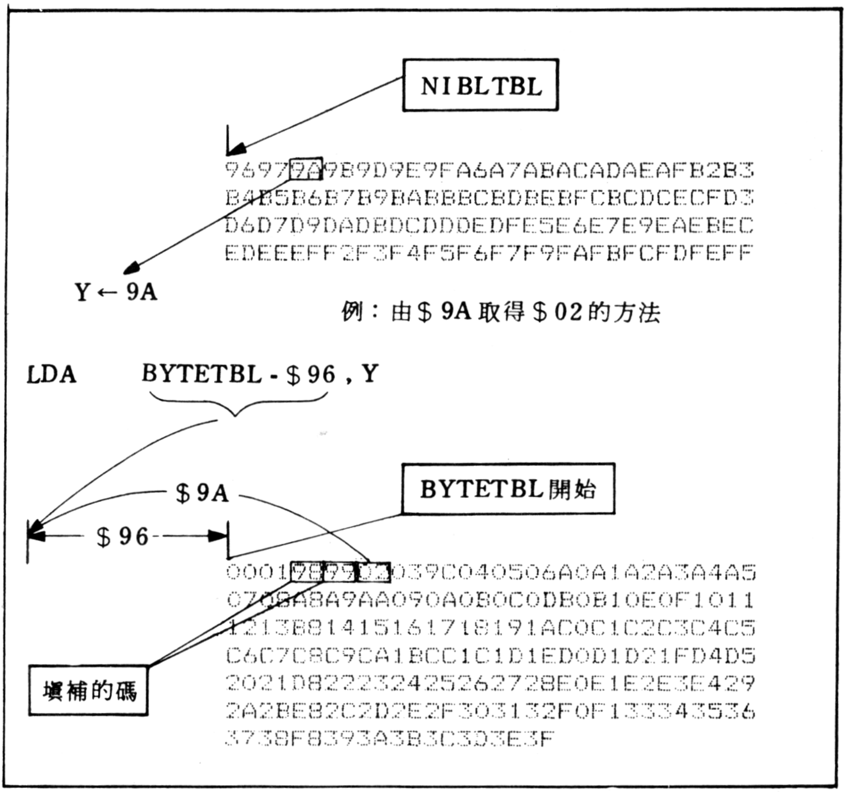


圖 6 － 7　查表還原法

所以程式第 206行的用意便很明白了，由磁片所讀到的是 WRITE 時查 NIBLTBL 寫入的碼，現在存入 Y 以做爲索引，然後用圖 6 — 7 的方法造出 BYTETBL 表並倒退 $96 個位元組來查取。206行除了查表之外，還兼有圖 6 — 6 的 EOR 還原功能，不過因爲位址索引的動作在先，而 EOR 的動作在後（如此才能作索引動作），所以是符合圖 6 — 6 的還原次序的。整個圖 6 — 6 的過程除了資料讀取之外，均在 206這行完成。現在，我們已完全明白 6 ／ 2 編碼、造編碼區（PRENIBBLE）、查表轉換、寫出資料、讀入資料、查表還原、 EOR 還原、解編碼區（POSTNIBBLE ）等等原理了。

6 — 4 RWTS主程式

現在，我們已準備齊全的可以進入 APPLE 眞正的磁碟 I ／ O 程式 RWTS 了。由於所有的基本動作均由前述的 READ _ BLOCK、WRITE _ BLOCK、PRE _ NIBBLE、 POST _ NIBBLE 以及第 0 章的 SEEKABS 、READADDR 等來完成，因此，RWTS 所做的只是如何使用這些基本程式來完成 FMS 所交待下來的工作，如此而已，並且這些工作多爲細碎如 ' 磁碟機是否正在運轉？' 或 ' 目前是幾號磁碟機被啓動？' 等等，所以程式之說明，大抵以在程式上加註解及各單元程式的描述方式行之。

閱讀本程式時，請參照 RWTS 使用表 IOB、DCT 表，由 38 行至 43 行的各個副程式均在本書介紹過，44 行的 MSWAIT 是磁頭移動時延時 100 μS 的副程式，而 45 行的 FORMAT 其

動作就如第0章所描述的 INIT 一樣。所以，RWTS 主程式是很容易解讀的。

RWTS：磁碟的區段輸出入程式

```
***************************
*   DOE 3.3 RWTS Routine   *
***************************
;
*-----------------------
*ZERO PAGE VARIABLES
*-----------------------
SCRATCH1  EPZ  $26
SCRATCH2  EPZ  $2A
CHKSUN    EPZ  $2C
SECTOR    EPZ  $2D
TRACK     EPZ  $2E
VOLUME    EPZ  $2F
DRIVNO    EPZ  $35
RWTSPTR   EPZ  $36
DCTPTR    EPZ  $3C
BUFFPTR   EPZ  $3E
MONTIME   EPZ  $46
IOBPTR    EPZ  $48
*-------------------
* DOS VARIABLES
*-------------------
CURTRK    EQU  $0478
SEEKCNT   EQU  $04F8
RTRYCNT   EQU  $0578
SLOT      EQU  $05F8
RECALCNT  EQU  $06F8
*-------------------
* DOS VECTORS
*-------------------
DRV0TRK   EQU  $0478     (～$47F)，磁軌#暫存區
DRV1TRK   EQU  $04F8     (～$4FF)，磁軌#暫存區
```

```
ILEAV     EQU $BFB8          (～$BFC7，磁區轉換表)
*--------------------
*OTHER DOS ROUTINES
*--------------------
PRENIBL   EQU $B800          ;程式6－0
WRITE     EQU $B82A          ;程式6－1
POSTNIBL  EQU $B8C2          ;程式6－2
READ      EQU $B8DC          ;程式6－2
RDADDR    EQU $B944          ;程式0－2
SEEKABS   EQU $B9A0          ;程式0－1
MSWAIT    EQU $BA00          ;等候100μS
FORMAT    EQU $BEAF          ; INIT
*------------------------
*DOS HARDWARE ADDRESSES
*------------------------
PHASEOFF  EQU $C080
PHASEON   EQU $C081
MOTOROFF  EQU $C088
MOTORON   EQU $C089
DRV0EN    EQU $C08A
DRV1EN    EQU $C08B
Q6L       EQU $C08C
Q6H       EQU $C08D
Q7L       EQU $C08E
Q7H       EQU $C08F
RWTS      ORG $BD00          ;程式開始
**********************************
* UPON ENTRY, STORE Y AND A AT   *
* $48,$49 AS POINTERS TO THE IOB*
*WRTS ROUTINE IS AT $B7B5.       *
**********************************
          STY IOBPTR         ;指向IOB所在
          STA IOBPTR+1
          LDY #2             ;最多可校正磁頭2次
          STY RECALCNT
          LDY #4             ;最多可搜尋磁軌4次
          STY SEEKCNT
          LDY ##$01
          LDA (IOBPTR),Y
```

```
               TAX                   ; X←新 slot #
               LDY #$0F              ; IOB+$0F=舊的 slot #
               CMP (IOBPTR),Y        ;與上次的 slot # 相同？
               BEQ SAMESLOT
*********************************
*UPDATE SLOT NUMBER IN IOB AND  *
* WAIT FOR OLD DRIVE TO TURN OFF*
*********************************
               TXA                   ;若不同，則更換 slot #
               PHA                   ;堆疊←新 slot #
               LDA (IOBPTR),Y        ;取舊的 slot #
               TAX                   ;做爲索引X
               PLA                   ; A←新 slot #
               PHA                   ; S（堆疊）指位器歸位
               STA (IOBPTR),Y        ; IOB←新 slot #
               LDA Q7L,X
STILLON        LDY #8
               LDA Q6L,X             ;讀資料
NOTSURE        CMP Q6L,X             ;若結果不變，表示磁碟機
               BNE STILLON           ;可能已停止運轉
               DEY                   ;若嘗試8次（≈200μS）
               BNE NOTSURE           ;均不變，表示OK
               PLA                   ;則取回新 slot #
               TAX                   ;做爲索引X
*************************************
* ENTER READ MODE AND READ WITH     *
* DELAYS TO SEE IF DISK IS SPINNING *
*************************************
SAMESLOT LDA Q7L,X                   ;設定磁頭成READ狀態
               LDA Q6L,X
               LDY #8
DRVSTART LDA Q6L,X                   ;讀一位元組
               PHA                   ⎫
               PLA                   ⎪
               PHA                   ⎬ 延時
               PLA                   ⎪
               STX SLOT              ⎭
               CMP Q6L,X             ;比較
               BNE DRVON             ;若訊號變化
```

```
                DEY
                BNE DRVSTART      ；再嘗試（最多8次）
DRVON           PHP               ；堆疊←狀態旗標群，其中
( 新 slot on )LDA MOTORON,X  ；Z=0表示馬達旋轉，Z=1停止
********************************
* MOVE POINTERS IN IOB TO 0-  *
* PAGE FOR FUTURE USE. CHECK  *
* DEVICE STATUS ....          *
********************************
                LDY #$06          ；IOB＋6＝DCT位址
PTRMOV          LDA (IOBPTR),Y    ；IOB＋8＝資料暫存區位址
                STA RWTSPTR,Y
                INY
                CPY #$0A
                BNE PTRMOV
                LDY #$03          ；DCT＋3＝時間常數
                LDA (DCTPTR),Y
                STA MONTIME+1
                LDY #$02          ；IOB＋2＝目前磁碟機＃
                LDA (IOBPTR),Y
                LDY #$10          ；IOB＋$10＝舊磁碟機＃
                CMP (IOBPTR),Y
                BEQ OK
                STA (IOBPTR),Y    ；若不相等，則更改之
**********************************
* SELECT APPROPRIATE DRIVE AND   *
* SAVE DRIVE BEING USED AS HIGH  *
* BIT OF DRIVNO (1=DRIVE1 ,      *
* 0=DRIVE2). GET TEST RESULTS.   *
**********************************
                PLP               ；將Z＝0存入狀態旗號群
                LDY #$00          ；Z＝0表示馬達已停止
                PHP
OK              ROR               ；將磁碟機號碼取入C
                BCC GDRV1         ；C＝0，D2；C＝1，D1
                LDA DRV0EN,X      ；啓動D1
                BCS DRVSEL        ；JMP DRVSEL
GDRV1           LDA DRV1EN,X      ；啓動D2
DRVSEL          ROR DRIVNO ；C→DRIVNO，+＝D2，-＝D1
```

```
                PLP                       ；取旗標，測試馬達狀態
                PHP
                BNE  MOTON                ；若 Z≠0，則馬達未停止
                LDY  #7
WAIT            JSR  MSWAIT               ；約等候 150 μS
                DEY
                BNE  WAIT
                LDX  SLOT                 ；取 slot #
*********************************
* GET DESTINATION TRACK AND GO TO*
* IT USING MYSEEK. CHECK TEST    *
* RESULT AGAIN AND IF DRIVE WAS  *
* ON, BRANCH TO TRYTRK. DELAY FOR*
* MOTOR TO COME UP TO SPEED.     *
*********************************
MOTON           LDY  #$04                 ； IOB + 4 ＝磁軌號碼
                LDA  (IOBPTR),Y
                JSR  MYSEEK               ；移動磁頭至所要的磁軌
                PLP                       ；測試旋轉狀態
                BNE  TRYTRK               ；若已旋轉，則不必延時
                LDY  MONTIME+1
                BPL  TRYTRK
MOTOFF          LDY  #18               ┐
CONWAIT         DEY                    │
                BNE  CONWAIT           │
                INC  MONTIME           ├ 約延時 1 秒，等候馬達
                BNE  MOTOFF            │ 旋轉至正確速度
                INC  MONTIME+1         │
                BNE  MOTOFF            ┘
*****************************
*     GET COMMAND CODE ...   *
*****************************
TRYTRK          LDY  #$0C                 ； IOB + $C ＝呼叫碼
                LDA  (IOBPTR),Y
                BEQ  GALLDONE             ；呼叫碼＝ 0 → do nothing
                CMP  #$04                 ；呼叫碼＝ 4 → INIT
                BEQ  FORMDSK
                ROR                       ； C＝1，read ； C＝0 write
                PHP                       ；將 C 狀態記入堆疊
```

```
               BCS TRYTRK2         ; if read，則不用編碼
               JSR PRENIBL
*********************************
* INITIALIZE MAXIMUM RETRIES TO *
* 48 AND READ AN ADDRESS FIELD  *
*********************************
TRYTRK2        LDY #48             ; 嘗試 48 = $ 30 次
               STY RTRYCNT
TRYADR         LDX SLOT
               JSR RDADDR          ; 讀目前位置
               BCC CHKTRK          ; 測試磁軌號碼
TRYADR2        DEC RTRYCNT
               BPL TRYADR
RECAL          LDA CURTRK          ; 磁頭歸零
               PHA                 ; 暫存磁軌號碼
               LDA #96             ; 設目前磁軌號碼 = 96
               JSR SETTRK
               DEC RECALCNT        ; 歸 0 計數，不可超過一次
               BEQ DRVERR          ; 否則 err
               LDA #4
               STA SEEKCNT         ; 磁軌尋找錯誤，最多 4 次
               LDA #0              ; Trk # = 0
               JSR MYSEEK          ; 歸 0
               PLA                 ; 取回所要的磁軌號碼
RESEK          JSR MYSEEK          ; 搜尋所要的磁軌
               JMP TRYTRK2
***************************
* VERIFY ON CORRECT TRACK.*
***************************
CHKTRK         LDY TRACK
               CPY CURTRK
               BEQ CHKVOL          ; OK，則檢查 vol #
               LDA CURTRK
               PHA
               TYA
               JSR SETTRK          ; 去設定磁軌號碼後
               PLA                 ; 再嘗試
               DEC SEEKCNT         ; 最多 4 次
```

```
               BNE  RESEK
               BEQ  RECAL
***********************************
* CLEAN UP STACK AND STATUS REG.  *
* LOAD A WITH $40 (DRIVE ERROR).  *
* GOTO HNDLERR.                   *
***********************************
DRVERR         PLA                     ;校正堆疊位置
               LDA  #$40               ; driver err
JMPTO1         PLP                     ;校正堆疊位置
               JMP  HNDLERR
GALLDONE  BEQ  ALLDONE                 ; C=0，則OK
FORMDSK   JMP  FORMAT
***********************************
* CHECK VOLUME NUMBER FOUND WITH  *
* VOLUME NUMBER WANTED. IF NO     *
* VOLUME WAS SPECIFIED, THEN NO   *
* ERROR. IF SPECIFIED VOLUME      *
* DOES NOT MATCH, LOAD A WITH $20*
* EXIT VIA HNDLERR.               *
***********************************
CHKVOL         LDY  #$03               ; IOB+3=所要找的vol #
               LDA  (IOBPTR),Y
               PHA                     ;暫存vol #
               LDA  VOLUME
               LDY  #$0E               ; IOB+$0E=眞正的vol #
               STA  (IOBPTR),Y
               PLA
               BEQ  CHKSECT            ;若vol #=0，OK
               CMP  VOLUME
               BEQ  CHKSECT            ;是否一樣
               LDA  #$20               ; $20是vol #不符的碼
               BNE  JMPTO1             ; JMP  JMPTO1
**************************
*    SECTOR CHECK ...    *
**************************
CHKSECT        LDY  #$05               ; IOB+$05=磁區號碼
               LDA  (IOBPTR),Y
```

```
              TAY                   ;以Y做索引
              LDA  ILEAV,Y          ;取對照表
              CMP  SECTOR
              BNE  TRYADR2          ; OK ?
              PLP                   ;取呼叫碼
              BCC  WRIT             ; C=0,write
              JSR  READ             ; C=1,read
              PHP                   ;將由read傳回來的C旗
              BCS  TRYADR2          ;號暫存,若C=1,則err
              PLP
              LDX  #0               ;準備POSTNIBL
              STX  SCRATCH1
              JSR  POSTNIBL
              LDX  SLOT
ALLDONE       CLC
              HEX  24               ; BIT指令,用以跳過SEC
********************************
* SET CARRY. STORE A IN IOB AS *
* RETURN CODE. TURN OFF MOTOR. *
* RETURN TO CALLER.            *
********************************
HNDLERR       SEC
              LDY  #$0D             ; IOB+$0D=err碼
              STA  (IOBPTR),Y
              LDA  MOTOROFF,X       ;停止馬達運轉
              RTS
*********************************
* WRITE A SECTOR USING WRITE.   (以下爲副程式群)
* IF BAD WRITE, LOAD A WITH $10 *
* (WRITE PROTECT ERR) ...       *
*********************************
WRIT          JSR  WRITE            ; write=$B82A
              BCC  ALLDONE          ; C=0,則OK
              LDA  #$10             ;否則write protected=$10
              BCS  HNDLERR
*********************************
* PROVIDES NECESSARY HOUSEKEEPING*
* BEFORE GOING TO SEEKABS.      *
```

```
* DETERMINES NUMBER OF PHASES     *
* PER TRACK AND STORES TRACK      *
* INFORMATION IN APPROPRIATE      *
* SLOT DEPENDENT LOCATION.        *
***********************************
MYSEEK   PHA                  ;暫存磁軌號碼
         LDY #$01             ;DCT+1=每條磁軌
         LDA (DCTPTR),Y       ;所對應的馬達相位數目
         ROR              ;C=0，半磁軌；C=1，全軌
         PLA                  ;取回號碼
         BCC SEEK             ;半軌
         ASL
         JSR SEEK             ;全軌
         LSR CURTRK           ;將SEEK改過的號碼還原
         RTS
SEEK     STA SCRATCH2         ;暫存號碼
         JSR XTOY             ;將插座號碼存入Y
         LDA DRVOTRK,Y        ;取D1的磁軌號碼
         BIT DRIVNO           ;D1=負，D2=正
         BMI WASDO
         LDA DRV1TRK,Y        ;取D2的磁軌號碼
WASDO    STA CURTRK           ;改CURTRK的值
         LDA SCRATCH2
         BIT DRIVNO           ;D1=負，D2=正
         BMI ISDRVO           ;D1
         STA DRV1TRK,Y        ;D2
         BPL GOSEEK
ISDRVO   STA DRVOTRK,Y
GOSEEK   JMP SEEKABS          ;開始搜尋磁軌
*********************************
* PUT SLOT IN Y BY TRANSFERRING*
* X/16 INTO Y                   *
*********************************
XTOY     TXA                  ;A←slot * 16
         LSR                  ⎫
         LSR                  ⎬ ;除以16
         LSR                  ⎪
         LSR                  ⎭
         TAY                  ;Y←slot
```

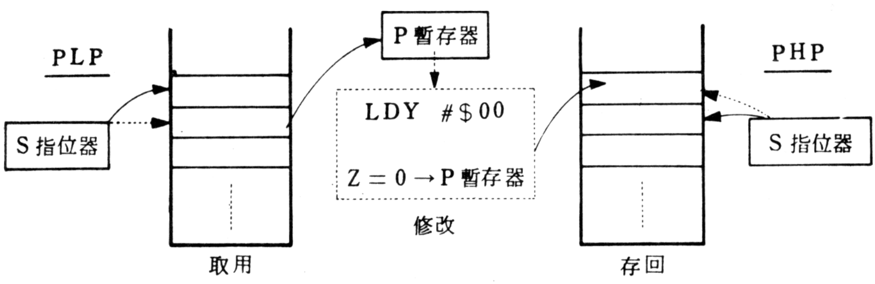
```
343             RTS
344 ********************
345 * SET TRACK NUMBER.*
346 ********************
347 SETTRK      PHA                  ; 暫存磁軌號碼
348             LDY #$02             ; IOB + 2 = 磁碟號碼
349             LDA (IOBPTR),Y
350             ROR
351             ROR DRIVNO           ; D1 = 負，D2 = 正
352             JSR XTOY             ; Y ← slot
353             PLA                  ; 取磁軌號碼
354             ASL                  ; 乘 2
355             BIT DRIVNO           ; D1 = 負，D2 = 正
356             BMI ONDRV0
357             STA DRV1TRK,Y
358             BPL SETRTS           ; JMP  SETRTS
359 ONDRV0      STA CURTRK,Y
360 SETRTS      RTS
361             END
```

程式 6 － 3　RWTS 主程式

RWTS 由第 59 行 $ BD00 開始，將 A 、Y 暫存器所含的值存入 IOB 指位器 ($ 48 , $ 49)，由於位址 $ 48 同時也被 APPLE 的監督程式做爲狀態旗標群暫存之用，因而結束 RWTS 時，要將 $ 48 設爲 0 ，否則十進位狀態旗標 D 會被設定爲 1 ，而使得 APPLE 工作失常 (譬如喇叭發出聲音) 。RWTS 首先檢查插座號碼 (S #) 是否正確 (75 行) ，若否，則由 81 至 96 行來更換 slot 號碼，其中 85、86 行是爲了使堆疊指位器 (S) 能夠指在原處，因而 PLA 之後馬上 PHA 還原之，其目的是在校正 S 所指的位置，本程式往後會有許多類似的情形。另外，

91 行的CMP　Q6L，X是用來檢知磁碟機是否仍在旋轉，若是，則必然陸續會讀到新資料，比較結果一定不相等，所以BNE STILLON，101至115行測試新插座的磁碟機是否轉動時也是採用這個方法，這是本程式內常見的方式。如果您要自行設計新DOS，RWTS 的許多技巧很值得參考。

檢查完插座號碼後，就由121至126行開始將 IOB 的DCT及資料暫存區的指位器內含值搬入 0 頁的（＄36，＄37）及（＄38，＄39），以做下一階段之用。127 至 129 行設定磁碟機的時間常數，接著便開始檢查磁碟機號碼（D#），142 至144行及151至152行再次以PLP、PHP來達成取用或修改堆疊器內容的目的：



如果磁碟機號碼已選好，並且馬達也運轉了，則以 JSR MYSEEK 來移動磁頭至目的磁軌。若馬達未運轉，則啓動馬達並等候 1 秒鐘使旋轉至正常速度。309行的MYSEEK 的作用是設定磁碟機做全軌或半軌讀寫，然後 SEEK，SEEK則依照前面所設定的插座、磁碟機、磁軌號碼來呼叫 SEEKABS，至於

SEEKABS的工作，則在程式0－1就已說明過了。

183行開始檢查呼叫碼，並依呼叫碼的類別分別至對應的副程式工作。196行的 TRYTRK2 設定尋找次數爲 48 （參考3－2節），然後以READDR（程式0－2）來讀（V，T，S）值，若動作無誤（C＝0），則至CHKTRK 檢視所讀到的（V，T，S）值。如果 RDADDR 48次均不成功（位置欄損壞），則以 RECAL 令磁頭歸0，再嘗試 48 次，在一切均不成功時，則至DRVERR 宣告失敗。

由 219行的 CHKTRK、249行的 CHKVOL、264 行的 CHKSEC 檢視了（V，T，S）值無誤之後，第270行以PLP來取出呼叫碼（現在只有C＝0：write及C＝1：read 兩種），並依呼叫碼至 WRIT（不是 WRITE ）或 READ 來結束 RWTS 的工作，當然READ或WRIT 是要用到程式6－0至6－2的各個副程式的。280至 291行的ALLDONE及HNDLERR是以C旗號來區別，281行的HEX 24是BIT 指令碼，這是一個塡補碼，程式若由ALLDONE進入，則281及287行被6502 CPU 看成BIT＄38而無任何作用，若程式由HNDLERR進入，則 287行的＄38 被當成SEC ，因而 err 訊息便由C旗號傳回原呼叫程式了。整個RWTS的工作便是這樣。

雖然RWTS 主程式是由＄BD00 開始，但呼叫時則必須由第3頁的向量或＄B7B5來進入。如果在不方便由上述方式進入RWTS的場合，最好也要自己寫一段呼叫RWTS的準備工作程式，茲將＄B7B5的 CALLRWTS 列出以供參考：

```
***********************
**  Caller of RWTS   **
***********************
RWTSENT   EQU $BD00
          ORG $B7B5
CALLRWTS  PHP              ;存P
          SEI              ; IRQ禁止
          JSR RWTSENT      ;至RWTS
          BCS ERR          ; C＝1 , err
          PLP
          CLC
          RTS
ERR       PLP
          SEC              ; err時，C＝1
          RTS
          END
```

程式 6 － 4　呼叫RWTS的例子

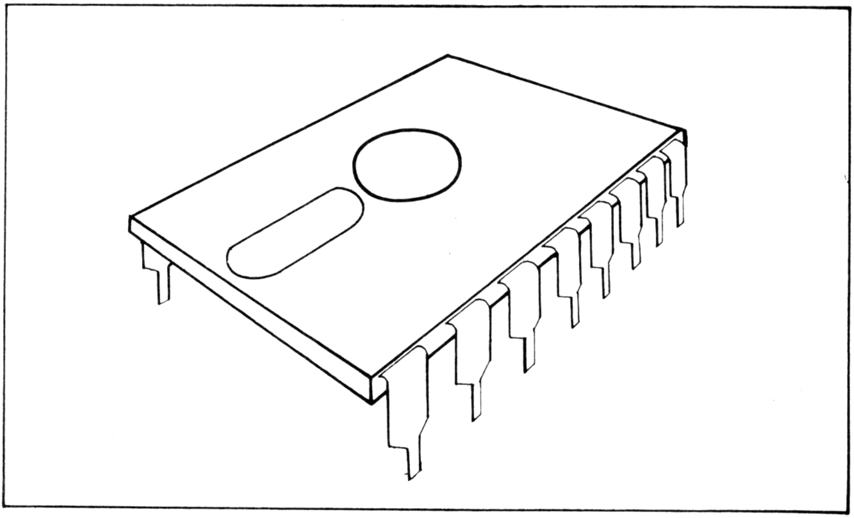
由前述我們知道RWTS使用了許多狀態旗號，因此爲了確保原來的旗號值，在呼叫RWTS前必須先將狀態旗號暫存器P的值記存起來。另外，爲了RWTS不被其他程式所中斷（ IRQ ），因而以 SEI 來將 IRQ 禁止。當JSR RWTSENT 返回後，若動作正確C旗號爲0，如果發生錯誤，則C旗號爲1，爲了確保C旗號不被 PLP 動作所掩蓋，因此 PLP 之後還要依C的狀態做 CLC 或 SEC 動作，這是時常引起迷惑的地方。

關於RWTS，要說的就是這些了。如果您有興趣進一步研習的話，則不妨以市面上各種加上保護的磁片來觀摩其RWTS改造的情形，通常RWTS並無太大改變，因此以尋找開頭指令：84 48 85 49或Q6 Q7 便很快可以判定RWTS的位置。

本章提要

在本書的最後一章，我們介紹您最快速的RAM--DISK，到底有多快呢？請執行看看…………

7 用RAM做成的磁片
——最快速的磁碟機——



7-0 最快速的磁碟機：RAM_DISK

我們都知道讀取檔案資料的時候，是先將含有該資料的磁區整個讀進資料暫存區（buffer），如果下次要讀寫的資料也是在這個磁區內，則只要讀寫資料暫存區而不必再讀寫磁片，顯然讀寫暫存區是比存取磁片快速多了。但是資料暫存區的空間畢竟有限（只有256位元組），如果我們有夠大的空間，是否可以將整塊磁片的資料全部讀進主機來呢？這樣存取速度可是最快的吧！

RAM卡的使用法

但是磁片的空間有143 K，APPLE 連 RAM 卡也才 64K*，因此只能每次取用一部份空間，此處選用 RAM 卡的16K 空間來使用。 RAM 卡雖然有16K 記憶體，但並非是由＄C000 至＄FFFF 連續排列，因為 APPLE 已將＄C000 至＄CFFF 的 4 K 位址做為 I ／O 用途，所以 RAM 卡的16K是這樣排列的：

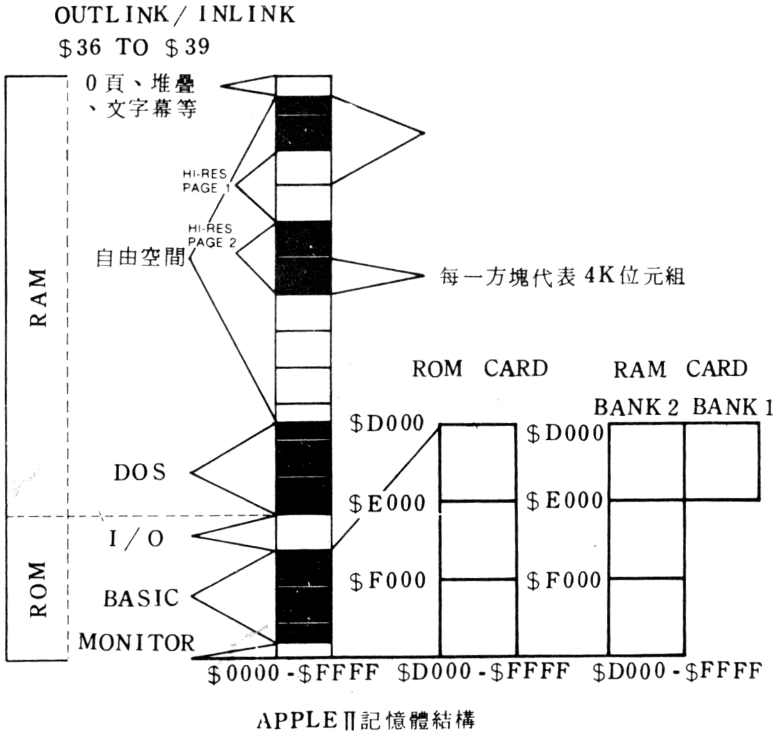


圖 7－0　APPLE的RAM卡位址

＊目前已有 256K RAM卡，其所附的ROM亦有類似的功能，本書以最常用的 16K 卡為主。

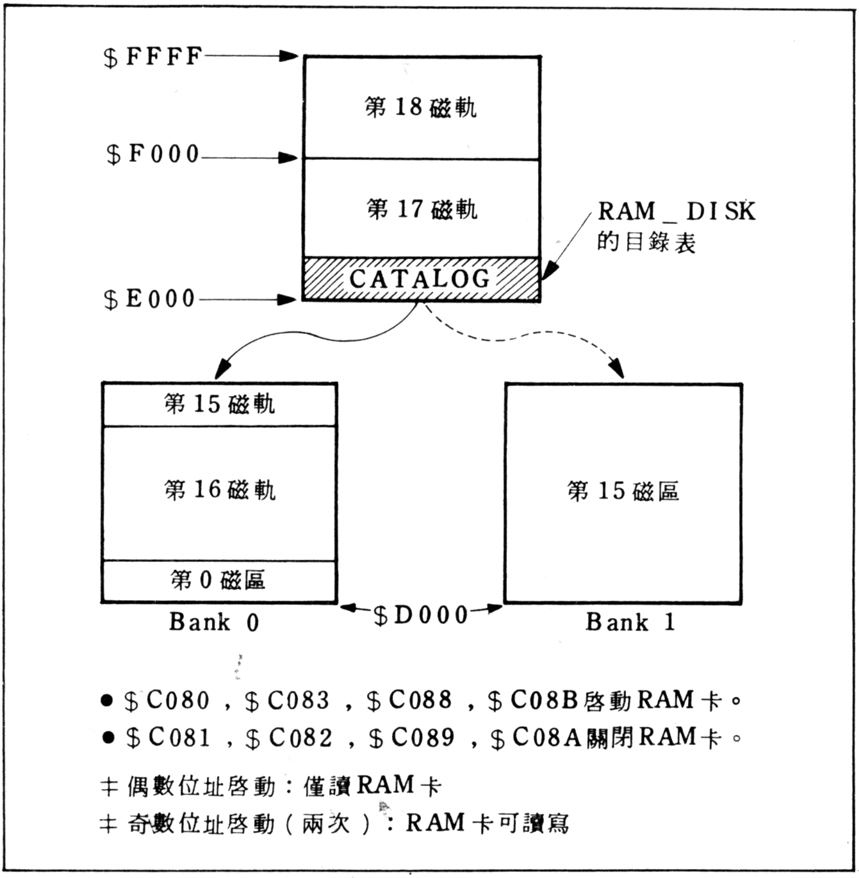


圖 7－1　以RAM卡來模擬磁碟機

由上圖我們可以看出，由 $D000 至 $DFFF 這 4 K位址上共有兩組 4 K的 RAM 佔用，但每次只能使用一組，二者啓動位址共距離 8 個位址，因此我們定義BANK這個變數值為 8（程式 7－0 第 69 行，兩個BANK的位址關係如下：

Bank 0 位址	功　　能	Bank 1 位址
$C080	僅讀RAM卡（執行）	$C088
$C081	關掉 RAM 卡程式，啓動 APPLE 的 ROM，但是觸動此位址兩次後，可將資料寫入 RAM卡。	$C089
$C082	關掉 RAM 卡程式，啓動 APPLE 的 ROM，且不可將資料寫入 RAM 卡。	$C08A
$C083	讀取（執行）RAM 卡，觸動此位址兩次才可以將資料寫入。	$C08B

此處我們選用$C083＝LCONWRT（Language Card ON ε Writable）及$C082＝LCOFFPRT（Language Card OFF ε write Protected），所謂ON 的意思就是可讀資料當然也可執行其內的程式，使用LCONWRT時必須觸動兩次才可寫入資料。並且由於兩個BANK 距離爲 8，因此程式中常使用 LDA LCONWRT，X 這種方式，若X＝0 則選用BANK 0及$E000～$FFFF，而X＝8時就選到BANK 1 及$E000～$FFFF。

RAM卡的劃分

由圖 7－1 我們可以看出，16K 的 RAM 卡每 4 K空間剛好對應到一個磁軌的 16 個磁區（256 位元組× 16＝4K），並且我們將之對應到第 15、16、17、18 磁軌。雖然第17 磁軌平時是用來存放VTOC 及目錄磁區，但是對於由 RAM 卡所模擬的小

型（ 4 軌）磁片，目錄表是不需要太大的，因此我們只將第 17 磁軌的 1 － 3 磁區當成目錄磁區，共可容納 21 個檔名（記得每個磁區可存 7 組 FDB ）。這 3 個磁區所對應的記憶位址由圖 7 － 1 可以看出是＄E100～＄E3FF，而＄E000～＄E0FF 是 VTOC。

7 －1 RAM_DISK程式說明

```
0800    ************************************
0800    *                                  *
0800    *           RAM........DISK        *
0800    *                                  *
0800    *          for 16k ram card        *
0800    *              EMULATOR            *
0800    *                                  *
0800    ************************************
0800    *
003C    A1L       EPZ $3C
003E    A2L       EPZ $3E
003E    BUFPTR    EPZ $3E
0040    FMNMPTR   EQU $40
0042    A4L       EPZ $42
0046    PTR       EPZ $46
0048    IOB       EPZ $48
0100    STACK     EQU $100
0478    TRKBYTE   EQU $478
0678    SLTBYTE   EQU $678
9AA6    TARGET    EQU $9AA6
9CD3    DOSBUF1   EQU $9CD3
9D00    DOS       EQU $9D00
9D84    COLDOS    EQU $9D84
9DBF    WRMDOS    EQU $9DBF
9F8B    DOSEXIT   EQU $9F8B
A2EA    CLOSE     EQU $A2EA
A56B    DOSDSBLE  EQU $A56B
A7D4    BUILDBUF  EQU $A7D4
```

```
AA57    MAXFILES EQU $AA57
AA5F    DOSCMD   EQU $AA5F
AAB1    MAXDEF   EQU $AAB1
AA65    KEYWRDS  EQU $AA65
AA75    FNAMEBUF EQU $AA75
B293    HOMETRK  EQU $B293
B39B    STKSAVE  EQU $B39B
B3BB    DOSVTOC  EQU $B3BB
B385    FMERRXIT EQU $B385
B5BB    PARMLIST EQU $B5BB
B5F8    FDRV     EQU $B5F8
B7B7    RWCALL   EQU $B7B7
B7E8    RWPARMS  EQU $B7E8
B7FB    DCT      EQU $B7FB
BD00    RWTS     EQU $BD00
BD2A    DRVCALL  EQU $BD2A
FDDA    PRBYTE   EQU $FDDA
FE2C    MOVE     EQU $FE2C
FF3A    BELL     EQU $FF3A
0800    *
001E    COUNT    EPZ $1E
001F    VTOCTRK  EPZ $1F
0800    *
0001    RWSLT    EQU 1
0002    RWDRV    EQU 2
0003    RWVOL    EQU 3
0004    RWTRK    EQU 4
0005    RWSCT    EQU 5
0008    RWBUF    EQU 8
000C    RWCMD    EQU 12
000D    RWRC     EQU 13
000E    RWVOLFND EQU 14
000F    RWSLTFND EQU 15
0010    RWDRVFND EQU 16
0800    *
E000    VTOC     EQU $E000
C080    LCONPRT  EQU $C080
C081    LCOFFWRT EQU $C081
C082    LCOFFPRT EQU $C082
C083    LCONWRT  EQU $C083
0008    BANK     EQU 8
0800    *
6000             ORG $6000
6000    *
6000    *
```

```
6000                *********************************
6000                *      PUT PATCHES INTO DOS     *
6000                * ONE PATCH HANDLES READ, WRITE *
6000                * AND INITIALIZE FUNCTIONS, THE *
6000                * OTHER ALLOWS EMU TO ABORT AN  *
6000                *  ATTEMPT TO TURN OFF A DRIVE  *
6000                *      WHICH DOES NOT EXIST     *
6000                *********************************
6000 A0 02          BEGIN    LDY #2
6002 B9 55 60       BGN1     LDA RWPTCH,Y
6005 99 B7 B7                STA RWCALL,Y
6008 88                      DEY
6009 10 F7                   BPL BGN1
600B A0 02                   LDY #2
600D B9 58 60       BGN2     LDA DRVPTCH,Y
6010 99 2A BD                STA DRVCALL,Y
6013 88                      DEY
6014 10 F7                   BPL BGN2
6016                *********************************
6016                * KEEP INIT FROM USING THIS DOS *
6016                *********************************
6016 A9 60                   LDA #$60          ;RTS
6018 8D 6B A5                STA DOSDSBLE
601B                *********************************
601B                *LET DOS USE TRK 17 FOR FILES   *
601B                *********************************
601B A9 0E                   LDA #14
601D 8D 93 B2                STA HOMETRK
6020                *********************************
6020                * MOVE EMU TO DOS FILE BUFFER   *
6020                *********************************
6020 A9 5B                   LDA #RELCODE
6022 85 3C                   STA A1L
6024 A9 60                   LDA /RELCODE
6026 85 3D                   STA A1L+1
6028 A9 A9                   LDA #RELCODE+LENGTH
602A 85 3E                   STA A2L
602C A9 62                   LDA /RELCODE+LENGTH
602E 85 3F                   STA A2L+1
6030 A9 A6                   LDA #TARGET
6032 85 42                   STA A4L
6034 A9 9A                   LDA /TARGET
6036 85 43                   STA A4L+1
6038 A0 00                   LDY #0
603A 20 2C FE                JSR MOVE
```

```
603D                 ************************************
603D                 * MAXFILES 2 GIVES MORE SPACE      *
603D                 ************************************
603D A9 02                      LDA #2
603F 8D 57 AA                   STA MAXFILES
6042                 ************************************
6042                 * TELL DOS TO REBUILD BUFFERS      *
6042                 ************************************
6042 A9 80                      LDA #TARGET-38
6044 8D 00 9D                   STA DOS
6047 A9 9A                      LDA /TARGET-38
6049 8D 01 9D                   STA DOS+1
604C 20 D4 A7                   JSR BUILDBUF
604F 20 53 9C                   JSR INIT0
6052 4C 7E 9B                   JMP EXIT
6055 20 A7 9A        RWPTCH     JSR EMU
6058 20 AF 9C        DRVPTCH    JSR DRVEMU
605B                 RELCODE    EQU *
605B                 ************************************
605B                 * LOCK BYTE :STORE A 0 HERE TO     *
605B                 * ALLOW SAVEING TO A NEW DISK      *
605B                 ************************************
9AA6                            ORG TARGET
9AA6                            OBJ $605B
9AA6 FF              LOCK       HEX FF
9AA7                 ************************************
9AA7                 * MAIN ENTRANCE TO EMU FROM        *
9AA7                 * PATCH AT BEGINNING OF RWTS       *
9AA7                 ************************************
9AA7                 * STORE POINTER TO IOB             *
9AA7                 ************************************
9AA7 84 48           EMU        STY IOB
9AA9 85 49                      STA IOB+1
9AAB                 ************************************
9AAB                 * CHECK FOR EMU'S NAME IN FILE     *
9AAB                 * MANAGER'S FILE NAME BUFFER       *
9AAB                 ************************************
9AAB A0 1D                      LDY #29
9AAD A2 19                      LDX #25
9AAF A9 A0                      LDA #$A0
9AB1 D9 75 AA        NXT1       CMP FNAMEBUF,Y
9AB4 D0 2C                      BNE CHKSLT
9AB6 88                         DEY
9AB7 CA                         DEX
9AB8 10 F7                      BPL NXT1
9ABA A0 02                      LDY #2
```

```
9ABC B9 B8 9C   NXT2     LDA EMUNAME,Y
9ABF D9 75 AA            CMP FNAMEBUF,Y
9AC2 D0 1E               BNE CHKSLT
9AC4 88                  DEY
9AC5 10 F5               BPL NXT2
9AC7            ***********************************
9AC7            *   IFEMU'S NAME IS PRESENT,      *
9AC7            *   CHECK FOR A VALID COMMAND     *
9AC7            *     0=INIT, 2=LOAD, 4=SAVE      *
9AC7            *       12=LOCK, 14=UNLOCK        *
9AC7            ***********************************
9AC7 AC 5F AA            LDY DOSCMD
9ACA C0 0C               CPY #12
9ACC F0 0B               BEQ LOCKEMU
9ACE C0 0E               CPY #14
9AD0 F0 07               BEQ LOCKEMU
9AD2 C0 05               CPY #5
9AD4 B0 0C               BCS CHKSLT
9AD6 4C 5B 9B            JMP EMUFNCT
9AD9            ***********************************
9AD9            *   LOCK AND UNLOCK FUNCTIONS     *
9AD9            ***********************************
9AD9 98         LOCKEMU  TYA
9ADA E9 0E               SBC #14
9ADC 8D A6 9A            STA LOCK
9ADF 4C 7E 9B            JMP EXIT
9AE2            ***********************************
9AE2            * PROCEDURE TO HANDLE COMMANDS    *
9AE2            *  WHICH DO NOT USE EMU'S NAME    *
9AE2            *  THese are recognized by the    *
9AE2            * fact that they use EMU's slot   *
9AE2            ***********************************
9AE2            *  Check for EMU's slot number    *
9AE2            ***********************************
9AE2 AD E9 B7   CHKSLT   LDA RWPARMS+RWSLT
9AE5 CD BB 9C            CMP SLOT
9AE8 D0 6E               BNE NOTEMU
9AEA            ***********************************
9AEA            *  Get command code and branch    *
9AEA            ***********************************
9AEA AD F4 B7            LDA RWPARMS+RWCMD
9AED F0 58               BEQ DONE
9AEF C9 04               CMP #4             ;FORMAT
9AF1 D0 03               BNE SVCMD
9AF3 4C 7B 9B            JMP INITEMU
```

```
9AF6 48           SVCMD    PHA                    ;SAVE CMD
9AF7              ***********************************
9AF7              * Make an allowed track number   *
9AF7              ***********************************
9AF7 AD EC B7              LDA RWPARMS+RWTRK
9AFA C9 13        HILMT    CMP #19
9AFC 90 04                 BCC LOLMT
9AFE E9 04                 SBC #4
9B00 B0 F8                 BCS HILMT              ;ALWAYS
9B02 C9 0F        LOLMT    CMP #15
9B04 B0 04                 BCS TRKOK
9B06 69 04                 ADC #4
9B08 90 F8                 BCC LOLMT              ;ALWAYS
9B0A 8D 78 04     TRKOK    STA TRKBYTE
9B0D E9 0F                 SBC #15
9B0F AA                    TAX
9B10              ***********************************
9B10              * Trnsfr buf pointer to page 0   *
9B10              ***********************************
9B10 AD F0 B7              LDA RWPARMS+RWBUF
9B13 85 3E                 STA BUFPTR
9B15 AD F1 B7              LDA RWPARMS+RWBUF+1
9B18 85 3F                 STA BUFPTR+1
9B1A              ***********************************
9B1A              *  Turn on card & select bank    *
9B1A              ***********************************
9B1A BC BC 9C              LDY TBL1,X
9B1D B9 83 C0              LDA LCONWRT,Y
9B20 B9 83 C0              LDA LCONWRT,Y
9B23              ***********************************
9B23              * Get byte equiv to track nmbr   *
9B23              *  and combine with sector to    *
9B23              *   make high byte of pointer    *
9B23              ***********************************
9B23 BD C0 9C              LDA TBL2,X
9B26 0D ED B7              ORA RWPARMS+RWSCT
9B29 85 47                 STA PTR+1
9B2B A9 00                 LDA #0
9B2D 85 46                 STA PTR
9B2F              ***********************************
9B2F              *  Recover command and branch    *
9B2F              ***********************************
9B2F 68                    PLA
9B30 4A                    LSR
9B31 B0 0B                 BCS READ
```

```
9B33 A0 00      WRITE    LDY #0
9B35 B1 3E      WR1      LDA (BUFPTR),Y
9B37 91 46               STA (PTR),Y
9B39 88                  DEY
9B3A D0 F9               BNE WR1
9B3C F0 09               BEQ DONE
9B3E            *
9B3E A0 00      READ     LDY #0
9B40 B1 46      RD1      LDA (PTR),Y
9B42 91 3E               STA (BUFPTR),Y
9B44 88                  DEY
9B45 D0 F9               BNE RD1
9B47 2C 82 C0   DONE     BIT LCOFFPRT
9B4A AD BB 9C            LDA SLOT
9B4D 8D F7 B7            STA RWPARMS+RWSLTFND
9B50 AD E0 9C            LDA VOLUME
9B53 8D F6 B7            STA RWPARMS+RWVOLFND
9B56 18                  CLC
9B57 60                  RTS
9B58 4C 04 BD   NOTEMU   JMP RWTS+4
9B5B            ***********************************
9B5B            *         Special functions       *
9B5B            *       (INIT, LOAD, and, SAVE    *
9B5B            *      to initialize EMU and to   *
9B5B            *      transfer EMU from/to disk  *
9B5B            *       0=INIT, 2=LOAD, 4=SAVE    *
9B5B            ***********************************
9B5B 98         EMUFNCT  TYA
9B5C D0 2A               BNE NOTINIT
9B5E            ***********************************
9B5E            * USE SLOT 5 IF NONE SPECIFIED    *
9B5E            ***********************************
9B5E A9 10               LDA #$10
9B60 2C 65 AA            BIT KEYWRDS
9B63 D0 04               BNE NOTDFLT
9B65 A9 50               LDA #5*16
9B67 D0 03               BNE STORE
9B69 AD E9 B7   NOTDFLT  LDA RWPARMS+RWSLT
9B6C 8D BB 9C   STORE    STA SLOT
9B6F AD EB B7            LDA RWPARMS+RWVOL
9B72 C9 FE               CMP #$FE
9B74 90 02               BCC SETVOL1
9B76 A9 00               LDA #0
9B78 8D E0 9C   SETVOL1  STA VOLUME
9B7B 20 53 9C   INITEMU  JSR INIT0
```

```
9B7E             **********************************
9B7E             * EXIT FOR INIT,LODA,AND  SAVE   *
9B7E             *                                *
9B7E             * (ZERO IOB LOCATION TO KEEP     *
9B7E             * MONITOR HAPPY AND RELEASE THE *
9B7E             *  FILE BUFFER OPENED FOR EMU    *
9B7E             *     BY THE FILE MANAGER        *
9B7E             **********************************
9B7E A9 00       EXIT     LDA #0
9B80 85 48                STA IOB
9B82 20 EA A2             JSR CLOSE
9B85 4C 8B 9F             JMP DOSEXIT
9B88             **********************************
9B88             * SAVE AND LOAD COMMANDS         *
9B88             *                                *
9B88             * STORE COMMAND FOR LATER        *
9B88             **********************************
9B88 4A          NOTINIT  LSR
9B89 8D ED 9C             STA EMUIOB+RWCMD
9B8C             **********************************
9B8C             * GET DISK AREA FOR LOAD/SAVE    *
9B8C             **********************************
9B8C AD 78 AA             LDA FNAMEBUF+3
9B8F C9 B1                CMP #"1"
9B91 90 08                BCC DFLT
9B93 C9 B9                CMP #"9"
9B95 B0 04                BCS DFLT
9B97 29 0F                AND #$0F
9B99 D0 02                BNE MAKETRK
9B9B A9 04       DFLT     LDA #4
9B9D             **********************************
9B9D             * CONVERT DISK AREA TO TRACK NO *
9B9D             **********************************
9B9D 0A          MAKETRK  ASL
9B9E 0A                   ASL
9B9F 18                   CLC
9BA0 69 01                ADC #1
9BA2 85 1F                STA VTOCTRK
9BA4             **********************************
9BA4             * IF LOAD, USE 0 FOR VOLUME      *
9BA4             **********************************
9BA4 AD ED 9C             LDA EMUIOB+RWCMD
9BA7 C9 01                CMP #1
9BA9 D0 04                BNE SETVOL2
9BAB A9 00                LDA #0
```

```
9BAD F0 03          BEQ SETVOL3
9BAF         **********************************
9BAF         * SET VOLUME EXPECTED IN IOB     *
9BAF         **********************************
9BAF AD E0 9C SETVOL2  LDA VOLUME
9BB2 8D E4 9C SETVOL3  STA EMUIOB+RWVOL
9BB5         **********************************
9BB5         * IF LOCK IS ON, ONLY SAVE YO    *
9BB5         *  A DISK WHICH EMU HAS MADE     *
9BB5         * (CHECK FOR NAME AND VOLUME     *
9BB5         **********************************
9BB5 2C A6 9A           BIT LOCK
9BB8 10 17              BPL SLTCHK
9BBA A5 1F              LDA VTOCTRK
9BBC 8D EC B7           STA RWPARMS+RWTRK
9BBF 20 04 BD           JSR RWTS+4
9BC2 B0 76              BCS RWERR
9BC4 A0 05              LDY #5
9BC6 B9 BB B3 IDLUP     LDA DOSVTOC,Y
9BC9 D9 DA 9C           CMP TBL5,Y
9BCC D0 60              BNE ERR
9BCE 88                 DEY
9BCF D0 F5              BNE IDLUP
9BD1         **********************************
9BD1         * CANNOT SAVE TO/LOAD FROM       *
9BD1         *       EMU'S OWN SLOT           *
9BD1         **********************************
9BD1 AD E9 B7 SLTCHK    LDA RWPARMS+RWSLT
9BD4 CD BB 9C           CMP SLOT
9BD7 F0 55              BEQ ERR
9BD9         **********************************
9BD9         * MOVE SLOT AND DRIVE NUMBERS    *
9BD9         *        TO SPECIAL IOB          *
9BD9         **********************************
9BD9 8D E2 9C           STA EMUIOB+RWSLT
9BDC AD EA B7           LDA RWPARMS+RWDRV
9BDF 8D E3 9C           STA EMUIOB+RWDRV
9BE2         **********************************
9BE2         * SET TRACK, LOW BYTE OF BUFFER  *
9BE2         *   POINTER, AND IOB POINTER     *
9BE2         **********************************
9BE2 A4 1F              LDY VTOCTRK
9BE4 C8                 INY
9BE5 8C E5 9C           STY EMUIOB+RWTRK
9BE8 A9 00              LDA #0
```

```
9BEA 8D E9 9C             STA EMUIOB+RWBUF
9BED A9 E1                LDA #EMUIOB
9BEF 85 48                STA IOB
9BF1 A9 9C                LDA /EMUIOB
9BF3 85 49                STA IOB+1
9BF5            ***********************************
9BF5            * READ/WRITE FOUR TRACKS          *
9BF5            ***********************************
9BF5 A9 03                LDA #3
9BF7 85 1E                STA COUNT
9BF9            ***********************************
9BF9            * CONVERT RELATIVE TRACK NUMBER *
9BF9            * (COUNT) TO BANK AND MEM RANGE *
9BF9            ***********************************
9BF9 A4 1E      RWLOOP    LDY COUNT
9BFB BE BC 9C             LDX TBL1,Y
9BFE BD 83 C0             LDA LCONWRT,X
9C01 BD 83 C0             LDA LCONWRT,X
9C04 B9 C0 9C             LDA TBL2,Y
9C07 09 0F                ORA #$0F
9C09 8D EA 9C             STA EMUIOB+RWBUF+1
9C0C            ***********************************
9C0C            * READ/WRITE 16 SECTORS           *
9C0C            ***********************************
9C0C A9 0F                LDA #$0F
9C0E 8D E6 9C             STA EMUIOB+RWSCT
9C11 20 04 BD   TRKLP     JSR RWTS+4
9C14 B0 24                BCS RWERR
9C16 CE EA 9C             DEC EMUIOB+RWBUF+1
9C19 CE E6 9C             DEC EMUIOB+RWSCT
9C1C 10 F3                BPL TRKLP
9C1E            ***********************************
9C1E            * READ/WRITE NEXT TRACK           *
9C1E            ***********************************
9C1E CE E5 9C             DEC EMUIOB+RWTRK
9C21 C6 1E                DEC COUNT
9C23 10 D4                BPL RWLOOP
9C25 AD EF 9C             LDA EMUIOB+RWVOLFND
9C28 8D E0 9C             STA VOLUME
9C2B 4C 34 9C             JMP END
9C2E 2C 82 C0   ERR       BIT LCOFFPRT
9C31 20 3A FF             JSR BELL
9C34 2C 82 C0   END       BIT LCOFFPRT
9C37 4C 7E 9B             JMP EXIT
9C3A 2C 82 C0   RWERR     BIT LCOFFPRT
```

```
9C3D A0 0D                    LDY #RWRC
9C3F B1 48                    LDA (IOB),Y
9C41 4A                       LSR
9C42 4A                       LSR
9C43 4A                       LSR
9C44 4A                       LSR
9C45 A0 03                    LDY #3
9C47 88             ERRLP     DEY
9C48 F0 03                    BEQ GTCODE
9C4A 4A                       LSR
9C4B 90 FA                    BCC ERRLP
9C4D B9 F2 9C       GTCODE    LDA ERRCODE,Y
9C50 4C 85 B3                 JMP FMERRXIT
9C53                **********************************
9C53                * THIS SECTION CLEARS THE CARD  *
9C53                *     CONSTRUCTS A VTOC AND A    *
9C53                *    THREE-SECTOR DIRECTORY AT   *
9C53                * $E000-4E3FF (TRK 17, SEC 0-3) *
9C53                **********************************
9C53                * ZERO THE ENTIRE LANGUAGE CARD *
9C53                **********************************
9C53 A9 FF          INIT0     LDA #$FF
9C55 85 47                    STA PTR+1
9C57 A0 00                    LDY #0
9C59 84 46                    STY PTR
9C5B 98             INIT1     TYA
9C5C 2C 83 C0                 BIT LCONWRT
9C5F 2C 83 C0       INIT2     BIT LCONWRT
9C62 91 46                    STA (PTR),Y
9C64 2C 8B C0                 BIT LCONWRT+BANK
9C67 91 46                    STA (PTR),Y
9C69 88                       DEY
9C6A D0 F3                    BNE INIT2
9C6C C6 47                    DEC PTR+1
9C6E A5 47                    LDA PTR+1
9C70 C9 D0                    CMP #$D0
9C72 B0 E7                    BCS INIT1
9C74                **********************************
9C74                * SET UP VTOC AT $E000          *
9C74                **********************************
9C74 A0 0D                    LDY #TBL4-TBL3+1
9C76 B9 C4 9C       INIT3     LDA TBL3,Y
9C79 99 74 E0                 STA VTOC+$74,Y
9C7C 88                       DEY
9C7D 10 F7                    BPL INIT3
```

```
9C7F A0 07                   LDY #TBL5-TBL4+1
9C81 B9 D2 9C     INIT4      LDA TBL4,Y
9C84 99 30 E0                STA VTOC+$30,Y
9C87 88                      DEY
9C88 10 F7                   BPL INIT4
9C8A A9 7A                   LDA #$7A
9C8C 8D 27 E0                STA VTOC+$27
9C8F A0 06                   LDY #ENDTBL-TBL5+1
9C91 B9 DA 9C     INIT5      LDA TBL5,Y
9C94 99 00 E0                STA VTOC,Y
9C97 88                      DEY
9C98 10 F7                   BPL INIT5
9C9A A9 11                   LDA #$11
9C9C A0 02                   LDY #$02
9C9E 8D 01 E3                STA VTOC+$301
9CA1 8C 02 E3                STY VTOC+$302
9CA4 88                      DEY
9CA5 8D 01 E2                STA VTOC+$201
9CA8 8C 02 E2                STY VTOC+$202
9CAB              ************************************
9CAB              * SHUT OFF CARD AND RETURN         *
9CAB              ************************************
9CAB 2C 82 C0                BIT LCOFFPRT
9CAE 60                      RTS
9CAF              ************************************
9CAF              * FOOL RWTS IF IT IS TRYING TO     *
9CAF              *  SHUT OFF DRIVE IN EMULATED      *
9CAF              *               SLOT
9CAF              ************************************
9CAF EC BB 9C     DRVEMU     CPX SLOT
9CB2 F0 03                   BEQ OUT
9CB4 DD 8C C0                CMP $C08C,X
9CB7 60           OUT        RTS
9CB8              ************************************
9CB8              *    CONSTANTS BY WHICH EMU        *
9CB8              *       IDENTIFIES ITSELF          *
9CB8              ************************************
9CB8 C5 CD D5     EMUNAME    ASC "EMU"
9CBB 50           SLOT       BYT 5*16
9CBC              *
9CBC 00 08 08     TBL1       BYT 0,BANK,BANK,BANK
9CBF 08
9CC0              *
9CC0 D0 D0 E0     TBL2       HEX D0D0E0F0
9CC3 F0
```

```
9CC4             523  *
9CC4 FF FF 00    524  TBL3     HEX FFFF0000FFFF0000
9CC7 00 FF FF
9CCA 00 00
9CCC FF F0 00    525           HEX FFF00000FFFF
9CCF 00 FF FF
9CD2             526  *
9CD2 11 01 00    527  TBL4     HEX 1101000023100001
9CD5 00 23 10
9CD8 00 01
9CDA             528  *
9CDA 04 11 03    529  TBL5     BYT $04,$11,$03,"E","M","U",212
9CDD C5 CD D5
9CE0 D4
9CE0             530  VOLUME   EQU *-1
9CE1             531  ENDTBL   EQU *
9CE1             532  *
9CE1 01 60 01    533  EMUIOB   HEX 016001001100
9CE4 00 11 00
9CE7 FB B7       534           ADR DCT
9CE9 00 00 00    535           HEX 0000000000000006001
9CEC 00 00 00
9CEF 00 60 01
9CF2 08 07 04    536  ERRCODE  BYT 8,7,4
024E             537  LENGTH   EQU *-EMU
9CF5             538           END

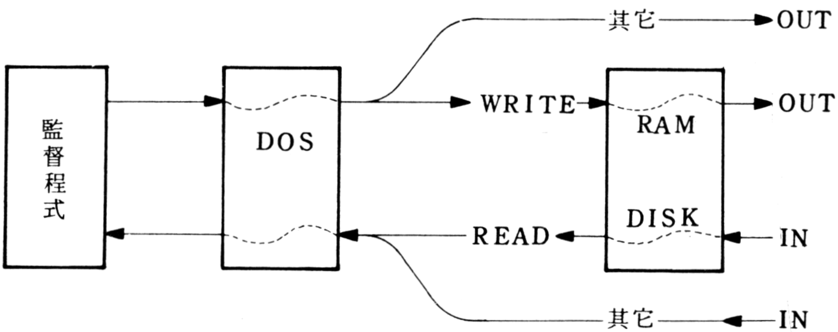

***** END OF ASSEMBLY
```

程式 7 － 0　RAM _ DISK

欄截DOS

本程式由＄6000開始執行　，首先對 DOS 的 RWCALL 及 DRVCALL 加以修改（ 82行至91行），使其分別指向 EMU 及 DRVEMU ，因爲RAM _ DISK 是模擬一個在第 5 號插座上的磁碟機，因此必須欄截有關 Driver 的訊息以檢查是否呼叫

RAM＿DISK ，另外讀寫動作也要攔截，以辨別是否爲RAM＿DISK（ S5 ）的讀寫而加以處理。第101行是設定起始磁軌爲 14 （因爲通常其值爲 17 ，且先往較 17 爲高的位置寫，然後再往較 17 低的位置寫，但 17 則不寫入資料，此處定爲 14 ，第 17 磁軌便可以用來寫資料了）。105至118行是將眞正的RAM＿DISK程式移往 DOS 的第一號資料暫存區，而第122至132行則重整並設定資料暫存區爲兩個（原來的第 2 、 3 號），如此一來RAM＿DISK程式便被夾在 DOS 與資料暫存區之間保護起來而不怕被重疊了。最後由133行的JMP EXIT 而結束整個RAM＿DISK的設定工作，並且將控制權交給 DOS ，再由 DOS 交給監督程式。



DOS 攔截監督程式　　　　RAM＿DISK 攔截 DOS

RAM＿DISK的設計構想與 DOS 對APPLE 輸出入的攔截相類似，但是其動作較爲單純，因爲所要處理的只有與磁片資料存取有關的指令而已，只要攔截檔案管理程式（ FMS ）的出口RWCALL（ $B7B7 ）便可以了，並且RAM＿DISK是假想成位

於第 5 號插座，因此欄截必須是命令中附有 S5 參數或 EMU 名稱的才發生作用，EMU 是 RAM 卡整體資料要存入眞正的磁片時規定的檔案名稱，必須使用這個檔名RAM__DISK 才會將 RAM 卡的資料存入磁片。插座號碼可以由 IOB 的SLOT #（$B7E9 ）取得，然後與第518 行的 SLOT（ $9CBB ）比較（199至 201行），至於EMU 這是給RAM__DISK專用的檔案名稱，RAM__DISK會檢視 FNAMEBUF（$AA75 ） 的內容看是否爲EMU（ 156 至 169 行）。

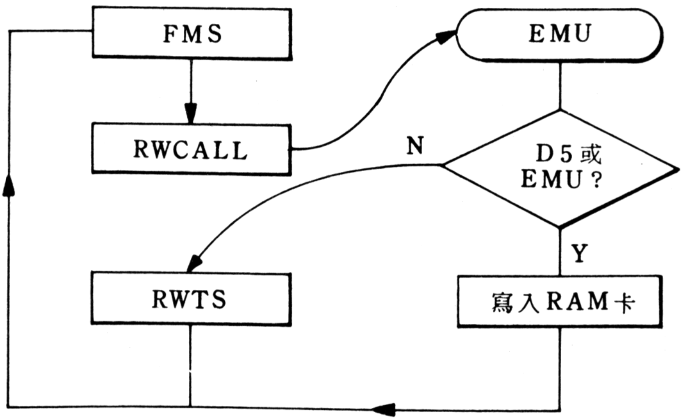


圖 7 － 2　RAM_DISK欄截流程

RAM__DISK這個程式因爲是模擬磁碟動作，因此必須在 RAM 卡上面執行 DOS 命令，譬如清洗RAM卡並且在$E000至$E3FF選出VTOC及目錄表的 INIT 命令、LOCK 、UNL-OCK命令等。並且當 RAM 卡的檔案處理完畢之後，也需要將它存入眞正的磁片以做永久保持之用（ LOAD EMU 及 SAVE

EMU）。所以RAM__DISK的功能可分成兩部份：①負責將資料暫存區的內容存入（取出）16K RAM 卡（模擬RWTS）。②負責將 RAM 卡的資料搬入磁片（呼叫眞正的 RWTS）。

模擬RWTS

RAM__DISK由RWCALL 處欄截DOS 是很聰明的方法，因爲許多DOS 命令實際上是由 FMS 所處理的，而到達RWTS時已經相當單純化了，觀察 IOB 的命令碼（第 0C 位元組）可以發現才只有0至3共4種情況要分別處理，而因爲RAM__DISK是沒有磁頭的磁碟機，因此根本不必要SEEK（命令碼爲0），所以在這個階層來欄截DOS RAM__DISK所要做的只有INIT、READ、WRITE 3個動作而已，比起在FMS 階層欄截（要做OPEN、WRITE、DELETE、……共12 種處理）是方便多了！

對RAM__DISK來說，要將資料暫存區的256 個位元組寫入（讀出）到 RAM 卡，其實是單純的記憶內容搬動而已，比起眞正的 RWTS 是簡單太多了！由199 行判定是 RAM 卡動作（S5）之後，再由207 行來判別是否 INIT 命令（命令碼＝4），若否，則暫時將命令碼暫存（PHA），接著由214 行至225 行來在 RAM 卡上找出可使用的磁軌。若 FMS 所指定的磁軌超過（少於）18（15），則減（加）4，一直到得出一個介於 15 至 18 的磁軌號碼，將這個號碼存入TRKBYTE（223行），減去15 後存入X暫存器以做爲Bank 選擇之用。

由229至232行是將資料暫存區起始位址存入BUFPTR（$3E,$3F）這個指位器內，以做爲將來搬動資料用。236至

238行是選出適當的 Bank ，其原則是只有當 FMS 指定使用第 15 磁區時（X之值爲0）才啓用Bank 0 ，否則均使用Bank 1 ，這可由 520行的TBL1配合X值做索引來選出（LDY TBL1 ,X 後Y之值爲0或8），然後LDA LCONWRT，Y 兩次便可選出適當的Bank 了。

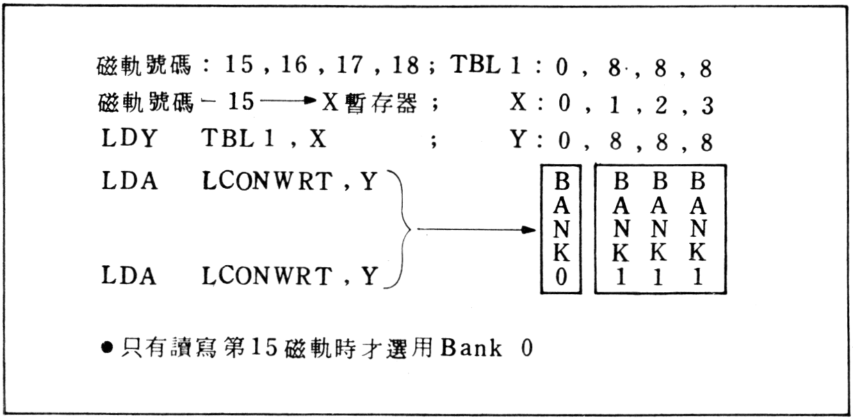


圖7－2　以磁軌號碼打開Bank的方法

由244行至248行是將磁軌號碼加上（ ORA ）磁區號碼後換成RAM 卡位址，最後存入PTR（＄46，＄47 ）指位器內，做爲資料移動之用。因爲各磁區的對應 RAM 卡起始位址其低位址均爲0 ：＄E000、＄E100、＄E200、……，因此 RTR （＄46）的內含値均爲0（247 行），而高位址則由TBL2依磁軌號碼來查出是D0、E0或F0，再與磁區號碼做 ORA 運算，便可以得出正確的值了。

```
          磁軌號碼  : 15,16,17,18
          X暫存器   :  0, 1, 2, 3
          TBL2      : D0,D0,E0,F0
LDA       TBL2,X    : D0,D0,E0,F0
ORA       RWPARMS + RWSCT : D0,D1,D2,……
          A(高位址)──►PTR+1 ╲
                              ──►D000,D100,……
         #0(低位址)──►PTR    ╱
```

圖7－3　以磁軌磁區號碼計算RAM卡位址的方法

一旦各起點、終點指位器均已設定完畢後，便可以依命令(現在 stack ，252行)是爲讀或寫來搬動資料了(253至 260行)。搬完資料之後當然還要關掉 RAM 卡(267 行)，並在IOB上設定其插座、磁片號碼(SLOT 、 VOL)爲 RAM卡的號碼，因此緊接著的命令若仍爲 RAM 卡命令，就不必再宣告S5 了，這樣便與一般DOS 的用法符合。272 行的 CLC 是用以告訴 FMS 讀寫成功(與RWTS一樣)，最後回到 FMS 。

如果 FMS 送過來的命令碼爲4 ，則RAM＿DISK要模擬RWTS做 INIT RAM卡的工作，也就是說清除 RAM 卡記憶內容並建立VTOC及目錄表。由456至471 行便是清除 RAM 卡的工作，而475 至498行則用以設定VTOC及目錄表，當 INIT工作完畢之後，由502行關閉 RAM 卡而回到 FMS 。VTOC的最初幾個位元組是＄04、＄11、＄03、“E”、“M”、“U”、212，由第2章的VTOC規則可以看出這些指的是目錄表由(＄11，＄03) 開始，而492至495行將(＄11，＄03)連至(＄11，＄02) ，496至498行又連至(＄11，＄01)而結束。

EMU 爲內藏的檔案名字，磁片號碼爲 212。TBL3 爲位元映射表，其中525 行一開始只有 3 個F 表示第 17 磁軌的 4 個磁區（1111 = F）已被用來做目錄表及VTOC 了。TBL 4 是表示目前所用磁軌爲 $ 11 ，接下來是加一號，磁碟共有 $ 23 磁軌，每磁軌共有 $10 個磁區，每個磁區有 $ 0100 位元組，與第 2 章介紹的情形一致。

將RAM卡資料存入磁片

剛才是RAM_DISK這個程式在模擬WRTS ，現在要將資料眞正的存入磁片 RAM_DISK 便無能爲力了，唯一的方法便是叫用眞正的RWTS ！

使用 RWTS 的方法我們是很熟悉的，只要設定 IOB 再呼叫 RWTS便可以了。由於 RAM 卡共有 16K = 64 磁區，因此 RWTS 呼叫共要 64 次（398 行至426 行），爲了使速度加快，我們不使用 DOS 的 IOB ，而另外由 533 行造了一個EMUIOB。至於計算及程式細節與模擬 RWTS 的部份類似，請參考程式註解。

由於每次 RAM 卡存入磁片要佔用 4 個磁軌，連磁片內DOS 佔用的磁軌算來，共能存放 8 個 RAM 卡檔案，我們將這些檔案名稱固定爲 EMU 1 、EMU 2 、……、EMU8 以免混亂：

檔案名稱	磁軌	目錄所在的磁軌
DOS	0 − 2	
EMU 1	3 − 6	5
EMU 2	7 − 10	9
EMU 3	11 − 14	13
EMU 4 (EMU)	15 − 18	17
EMU 5	19 − 22	21
EMU 6	23 − 26	25
EMU 7	27 − 30	29
EMU 8	31 − 34	33

圖 7 − 4　各檔案的磁片保留區

7 −2 RAM_DISK的使用法

如果你有磁片，請直接BRUN RAM_DISK.OBJ，否則較快的方法是將程式 7 − 0 的機械碼直接鍵入 APPLE，然後 BSAVE RAM_DISK.OBJ，A$6000，L$2AB，在鍵入過程當中要注意的是第140至141行之間不必將位址由$605B改成$9AA6，只要直接下去（$605B、(D、……）便可以了，因爲104至118行的程式會自動將它移動至正確的位置去，這是因爲本程式是以LISA組譯器發展出來的，而LISA（64K系統）的可用空間爲$800～$8000，因此第141行的ORG TARGET（$9AA6）之後要緊跟著OBJ $605B，然後用104～118行的程式來搬動。同樣的理由，本程式只適用於正常DOS

系統，至於其它系統（如LISA）的場合，必須注意其空間的相容性。

現在請做下列試驗：

```
LOAD   HELLO
SAVE   HELLO,S5
NEW
LIST
RUN    HELLO
CATALOG
結果？
```

毋庸置言，RAM_DISK的速度是快速無比的！除非您將命令參數改成S6，否則現在 APPLE 便一直在RAM_DISK狀態下。RAM_DISK也一樣可以在BASIC 程式當中使用：

```
10  PRINT  CHR$(4);" CATALOG,S5 "
RUN
```

這種功能使得在讀寫文字檔時更大大的顯示出RAM_DISK優越的速度。

特殊的EMU命令

EMU 是RAM_DISK系統一個特殊的保留字，一般的檔案不可以此爲檔名。如果 EMU 被放在 INIT、SAVE、LOAD、

LOCK、UNLOCK 之後，這些命令便被當成特殊命令來處理：

INIT EMU

在 INIT 磁片的時候，存在於主機內部的 DOS 程式會被存到此新磁片的 0 至 2 號磁軌，但是由於 RAM_DISK 程式已對 DOS 做了修改（RWCALL 等），並且當磁片啓動時 RAM_DISK 並不自動執行，如此將使已經修改的 DOS 不能工作（RWTS 斷掉了），因此當 RAM_DISK 執行後，INIT HELLO 這個命令只對磁片做規格化動作（Formatting）而不將 DOS 拷貝到磁片上。

INIT EMU 可以用來設定 RAM_DISK 所在插座，譬如 INIT EMU，S2 便可以將 RAM_DISK 設置於第 2 號插座，若改成 S3 則 RAM_DISK 便又改置於第 3 號插座了，所有這些改變只是改變命令參數 S 的值，RAM_DISK 真正是位於 $9AA6 而不在任何插座上。雖然在前一節的程式中，RAM 卡的磁片號碼被 TBL5 設成 212，但使用 INIT EMU，V××× 可以將磁片號碼更改。使用 INIT EMU 時要特別小心按鍵，譬如 INIT EEMU 會使 DOS 當成正常命令而將磁碟機內的磁片資料消掉！

SAVE EMU 及 LOAD EMU

執行 RAM_DISK 後，SAVE 及 LOAD 共有 3 種不同的正確執行情況：

```
SAVE HELLO,S6
SAVE HELLO,S5
SAVE EMU3
```

第一種是正常的DOS SAVE，第二種是將主機內的BASIC 程式以HELLO 爲檔名存入RAM 卡，而第3 種是將 RAM 卡的內容以 EMU3 爲檔名存入磁片，EMU 及EMU1、EMU2、……EMU8爲特殊的保留檔名，它們會分別被存到磁片上固定的位置上（圖 7 － 4 ）。用來存放 RAM 卡資料的磁片與一般有所不同，必須用下列的程式來規劃：

```
0   REM ***********************************
1   REM *    PREPARE THE DISK FOR EMU...    *
2   REM ***********************************
100  TEXT : HOME
120  IF ( PEEK (47032) <  > 167) THEN  PRINT
      CHR$ (4)"BRUN RAM_DISK.OBJ"
130  PRINT : FLASH : PRINT "WARNING: ";: NORMAL
     : PRINT " THIS PROGRAM WILL DESTROY "
     : PRINT "          ANYTHING ON THE DI
     SK!!!"
140  PRINT "PUT AN EMPTY DISK INTO THE DRI
     VE ..."
150  PRINT "IF OK HIT RETURN": GET A$: PRINT
     : IF A$ <  >  CHR$ (13) THEN 300
160 D$ =  CHR$ (4): PRINT D$"INIT EMU": PRINT
     D$"INIT HELLO,S6": PRINT D$"UNLOCKEMU
     "
200  FOR EMU = 1 TO 8: PRINT D$"SAVEEMU"EM
     U: NEXT EMU: PRINT D$"LOCKEMU": HOME
     : VTAB 8: PRINT "PREPARE ANOTHER DISK
```

```
        ?";: GET A$: IF A$ =  CHR$ (89) THEN
        140
300  HOME : PRINT "GOODBYE...": END
```

程式 7－1　製造RAM卡磁片的程式

使用這個程式造出來的磁片才可以使各個 EMU 程式固定在每個位置上。程式中第200行的 FOR＿NEXT 迴圈做的便是檔案定位的工作。

可是雖然在磁片上規劃出EMU1～EMU8 但CATALOG後是看不到檔名的。這個磁片的用法是每次將 RAM 卡的16K資料以EMU1～EMU8中的一個檔名存入磁片，但因爲只有EMU4（EMU）的位置與VTOC及目錄表重合，因而CATALOG，S6時只有 EMU4 的內容看得到，其他的 EMU 雖已SAVE到磁片但却不顯示出來。

```
SAVE          TEST,S5
SAVE          TEST1             將檔案存入RAM卡
SAVE          TEST2
SAVE          EMU,S6            將RAM卡存入磁片
INITEMU       ATEST,S5          清除RAM卡
SAVE          ATEST1
SAVE          ATEST2            將檔案再存入RAM卡
SAVE          EMU3,S6
SAVE          EMU,S6            將RAM卡存入磁片
LOAD          EMU3,S6
CATALOG,S5
LOAD
CATALOG,S5
..................
```

LOCK EMU與UNLOCK EMU

普通磁片不能存放 EMU 的原因是EMU（EMU 4）會破壞磁片上 17 號磁軌上的目錄表，爲了避免不小心的錯誤，RAM_DISK有了特殊的防護處理。首先它會檢查磁片VTOC上的3－5位元組，看是否內藏有EMU 三個字，若否，則不是EMU專用磁片不可SAVE。另外，RAM_DISK還會檢查磁片號碼（Vol #）看是否相同，若不同，也不能SAVE或LOAD。

使用UNLOCK EMU 可以令RAM_DISK放棄檢查磁片上是否有EMU 這3個字，因此UNLOCK 後，任何磁片只要磁片號碼相同便可以用來存放 RAM 卡的資料了。LOCK 則是用以恢復磁片檢查的功能。在程式7－1的第160行便是使用UNLOCK EMU 來使磁片能夠存入INIT後的 RAM 卡資料，因爲在還沒存過 RAM 卡資料的磁片是不會內藏 EMU 這3個字的，這是UNLOCK 的主要用途。

7－3 這樣的磁碟機是否快速？

RAM_DISK最大的優點是其極快的速度，而這個優點在文字檔讀寫的場合尤其有用。APPLE 在應用上最重要的是其資料處理的能力，而這個功能全靠文字檔讀寫來完成，可是磁片讀寫的速度是相當慢的，對於一個實用程式而言，往往需費時數分鐘至數小時，這種情形以RAM_DISK來處理困難便可解決了。

或許您會認爲，RAM_DISK的結果最後還是要存入磁片，如此速度有所改善嗎？其實檔案處理的問題並不在於最後的存取

時間，而是在處理過程當中Ｉ／Ｏ所造成的延時，譬如平均每次讀寫一筆資料的速度為１秒，若在處理過程當中需要讀寫500次，則便耗費了500秒，若以RAM＿DISK來處理，大概只要５秒吧！下列的程式可以用來證明RAM＿DISK的實力：

```
0   REM ***********************************
1   REM *     RAM_DISK DEMO ..........    *
2   REM ***********************************
120  IF ( PEEK (47032) <  > 167) THEN  PRINT
      CHR$ (4)"BRUN RAM_DISK.OBJ"
130 D$ =  CHR$ (4): DIM DR$(1):DR$(1) = "
     > EMU <":DR$(0) = "> DISK <"
140  FOR DR = 0 TO 1:SL = 6 - DR: HOME : VTAB
     2: PRINT : VTAB 8: PRINT "BUILDING A
      TEXTFILE ON "DR$(DR): GOSUB 190: NEXT
     DR
150  HOME : VTAB 4: PRINT "TYPE A NUMBER
     FROM 1 TO 26": PRINT : INPUT "0 WILL
      END THE PROGRAM  ";RC: ON RC = 0 GOTO
     180: ON RC < 1 OR RC > 26 GOTO 150
160  FOR DR = 0 TO 1: PRINT : PRINT  CHR$
     (7)"RETRIEVING RECORD "RC" FROM "DR$
     (DR): GOSUB 200: PRINT  CHR$ (7)RC$:
      PRINT "ANY KEY TO CONTINUE  ";: GET
     A$: PRINT : PRINT : NEXT DR
170  PRINT "SEE THE DIFFERENCE?": FOR I =
     1 TO 1000: NEXT I: GOTO 150
180  HOME : PRINT "GOODBYE": PRINT D$"DEL
     ETETEXT,S6": END
190  PRINT D$"OPENTEXT,L20,S"SL: FOR I =
     1 TO 26: PRINT D$"WRITETEXT,R"I:C =
     I + 64: PRINT  CHR$ (C) CHR$ (C)"  "
     DR$(DR)"  " CHR$ (C) CHR$ (C): NEXT
```

```
     I: PRINT D$"CLOSE": RETURN
200 SL = 6 - DR: PRINT D$"OPENTEXT,L20,S"
     SL: PRINT D$"READTEXT,R"RC: INPUT RC
     $: PRINT D$"CLOSE": RETURN
```

錯誤情形

使用RAM_DISK時若發生錯誤，喇叭會發出響聲。常見的錯誤情形是：

1. 使用SAVE FILE,S5 之後SAVE EMU。這是因爲第一次SAVE之後插座便設定成第5號，而SAVE EMU便將RAM卡的內容SAVE到RAM卡上，這是不被允許的。
2. SAVE或LOAD EMU普通磁片。對普通磁片存取EMU檔案要先UNLOCK EMU才行。
3. 磁片號碼不符合。
4. NO BUFFERS AVAILABLE。因爲RAM_DISK程式佔用了一個DOS的檔案資料暫存區，所以可用的資料暫存區只有兩個，遇到這種情況，可以在BRUN RAM_DISK之後（還未執行程式時）MAXFILES 3 便可解決。

結束RAM-DISK

由於RAM_DISK修改了DOS，因此結束RAM_DISK時，同時也要將DOS修正，你可以關機（或PR#6）重新BOOT DOS，但若要保留主機內的程式及資料，可以用下列的程式來脫離RAM DISK及修正DOS：

```
!L
;
***********************************
*    DISCONNECT RAM_DISK .....    *
***********************************
DOSBUF1   EQU $9CD3
DOS       EQU $9D00
DOSEXIT   EQU $9F8B
CLOSE     EQU $A2EA
DOSDSBLE  EQU $A56B
MAKEBUF   EQU $A7D4
FMNMBUF   EQU $AA75
MAXDEF    EQU $AAB1
HOMETRK   EQU $B293
RWCALL    EQU $B7B7
DRVCALL   EQU $BD2A
LCOFFPRT  EQU $C082
***********************************
          ORG $280
***********************************
* DISCONNECT EMU & RESTORE DOS    *
***********************************
RESTORE   LDY #2
RST1      LDA RWRSTR,Y
          STA RWCALL,Y
          DEY
          BPL RST1
          LDY #2
RST2      LDA DRVRSTR,Y
          STA DRVCALL,Y
          DEY
          BPL RST2
          LDA #$4C              ;JMP
          STA DOSDSBLE
***********************************
*  RESTORE TRACK SEEK PROTOCOL    *
***********************************
          LDA #17
```

```
                STA HOMETRK
*********************************
*       MAKE SURE CARD IS OFF     *
*********************************
                LDA LCOFFPRT
*********************************
*      CLOSE ALL OPEN FILES       *
*********************************
                LDA #$A0
                STA FMNMBUF
                JSR CLOSE
*********************************
*        RESTORE MAXFILES 3       *
*********************************
                LDA #3
                STA MAXDEF
*********************************
*   TELL DOS TO REBUILD BUFFERS   *
*********************************
                LDA #DOSBUF1
                STA DOS
                LDA /DOSBUF1
                STA DOS+1
                JSR MAKEBUF
                JMP DOSEXIT
RWRSTR          HEX 2000BD
DRVRSTR         HEX DD8CC0
                END
```

```
!BR
*280.2C5

0280- A0 02 B9 C0 02 99 B7 B7
0288- 88 10 F7 A0 02 B9 C3 02
0290- 99 2A BD 88 10 F7 10 4C
0298- 8D 6B A5 A9 11 8D 93 B2
02A0- AD 82 C0 A9 A0 8D 75 AA
02A8- 20 EA A2 A9 03 8D B1 AA
```

```
02B0- A9 D3 8D 00 9D A9 9C 8D
02B8- 01 9D 20 D4 A7 4C 8B 9F
02C0- 20 00 BD DD 8C C0
```

参考文獻：

Ref 1. RAM DISK, Ken and F.Manly, Nibble Vol 4 No 8, Dec 1983

A 磁片記錄的基本原理

單、雙密度磁片的原理
—— 各種技術的比較 ——

參考資料：

1. 美國聯邦專利 3108261 號：Recording and / or Reproducing Svstem
2. 美國聯邦專利 3560947 號：Method and Apparatus for Communication and Storage
3. Proposed American National Standard Recorded Magnetic Tape for Information Interchange (6250 CPI , GCR)

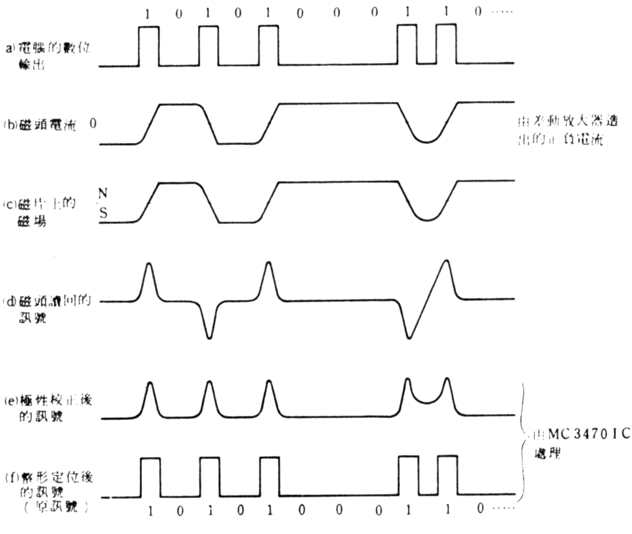
A－0 在磁片上記錄資料的方法

由第 0 章圖 0 － 2 及 0 － 3 可以知道資料的讀、寫是以磁頭

產生磁場來將磁片上面的磁性材料感磁*，這樣便在磁片上造出一個記號（感磁點）來，下次磁頭經過這個位置（在讀的狀態下）時，因爲受到這個感磁點的感應，便可以查覺出此處曾經記有一個資料，這便是磁碟機最基本的訊息存、取方法。

直接記錄法：NRZ(NO RETURN ON ZERO)

所謂直接記錄法是，每次電腦輸出一個脈衝（即1訊號），便令磁頭電流發生變化，譬如是由正變負或由負變正。一旦磁頭電流變化，其在磁片上所感應的磁場亦發生變化，磁場變化會造成磁通量變化，而磁頭便是利用這種磁通量的變化而讀回訊號的

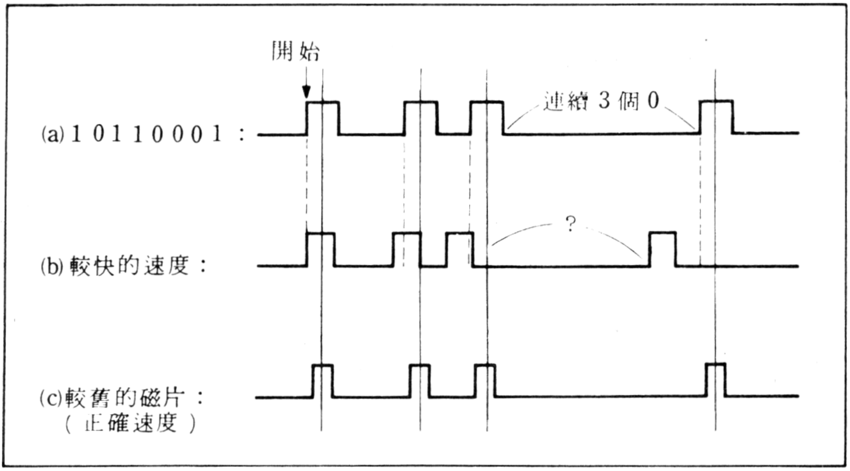


圖A－0 NRZ資料記錄法

---

*此處所謂的感磁，指的是磁通量變化，對於磁通量變化的點，磁頭可以感應出訊號（譬如'1'）來。至於感應不出訊號的點，並不一定沒有感應磁場，只是磁通量沒有變化而已（譬如磁飽和）。可參考電磁理論相關書籍，此點並不影響往後的討論。

直接記錄法是最原始也最有效率的磁片記錄法，這個方法每次遇到一個1便在磁片上造出一個感磁點（磁場變化），而對於0訊號則造出一飽和點（電流維持一個方向不變），這種方法使得0訊號得以將該位置上舊的1訊號清除（反之亦然）。由於直接記錄法的磁場只在1訊號時發生變化，而對於0訊號則不做任何反應，也就是說磁場（電流）並不因0訊號而改變方位（比較上圖(a)與(c)），所以稱爲No Return on Zero簡稱 NRZ 。



圖A－1　NRZ的缺點

NRZ 的缺點是如果磁碟機速度變化，則0訊號便很難判定了，譬如圖A－1(b)是速度變快的場合，原來由3個連續0訊號佔用的位置現在變窄了，電腦便難以判斷到底是2或3個0訊號（介於2～3之間時），如此便發生讀取錯誤。或許您會想到，雖然0訊號所代表的位置變窄了，但代表1訊號的位置不也變窄了嗎？我們如果用圖A－0(b)1訊號的寬度來量0訊號的寬度，

則仍然可以知道它是由 3 個 0 訊號所構成的。問題是，當磁片使用一段時間之後，其所記錄的訊號會發生退磁現象：

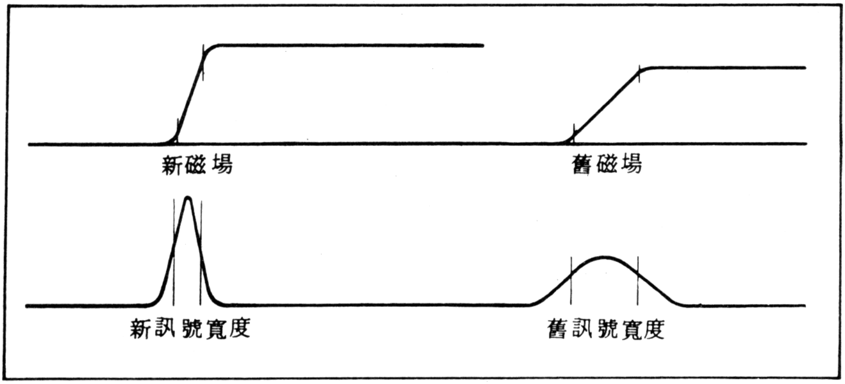


圖 A－2　訊號寬度的變化

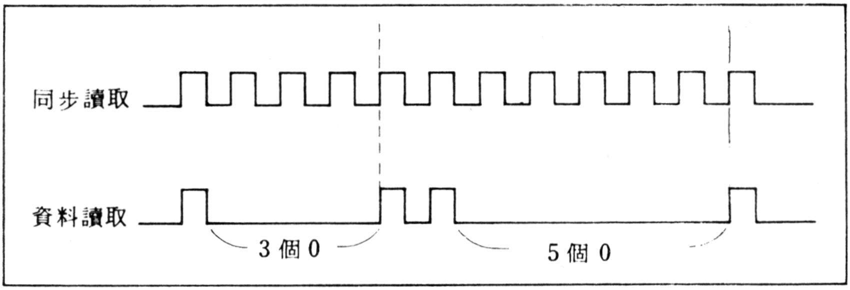
由上圖我們可以發現，即使磁碟機速度正確，但因爲退磁現象，訊號寬度也是會改變，並且寬度（時距）測量也是有所困難的，這便是我們不使用 NRZ 的原因*。

自同步記錄法：FM

NRZ 的困難是在於無法準確算出連續 0 訊號的個數（ 1 訊號是沒問題的）。對於多磁頭的磁碟機，可以在一特殊磁軌上錄上一同步脈衝（ clock ），並且專門用一個磁頭來讀此脈衝，以做爲其它磁頭訊號計數的參考，這樣便不會有速度偏移的問題了：

---

* 不過，下列介紹的 FM、MFM、GCR、……均爲 NRZ 的改良，因而眞正的磁片寫入動作仍舊是 NRZ 。



圖A－3　多磁頭的同步訊號

但是微電腦的磁碟機只有一個磁頭，同步訊號要怎麼記錄呢？解決的辦法是將同步訊號與資料訊號交互間隔來排列，也就是任何資料訊號的前後必須由同步訊號來做間隔。這樣子，在同步訊號間一定只有一個資料位元　當然沒有相鄰的0訊號，也就不會發生不易分辨的錯誤了。可是要怎麼分辨同步訊號與資料呢？其實很簡單，因爲0訊號的下個訊號便是同步訊號了。這種方法因爲在資料間加入同步訊號，因此稱爲FM方式（調頻），FM方式基本上仍是採用 NRZ 的訊號記錄法，只是加入同步訊號而使得每個資料位元之間得以隔離而解決了原始 NRZ 的困難。不過由於同步訊號的加入，使得磁片的空間浪費了一半，這便是一般所稱的單密度記錄法（single density），如果要在一片磁片上記錄同樣數量的資料，則頻率便得加倍（寬度減半），所以又稱爲DF（Double Frequency）方式。

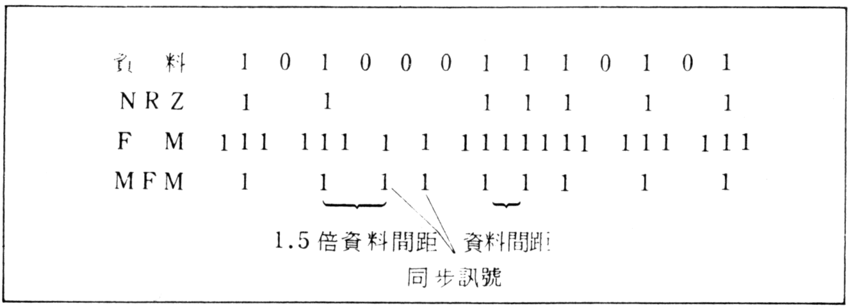
A－1 雙倍密度記錄法

由於磁性材料之間互相具有吸引及排斥力（其實是磁場作用

力），太靠近的訊號會相互影響而造成訊號移位的現象並且降低訊號電平，並且由於硬體上的限制，太密集的資料會造成讀取困難。因此要增加磁片資料密度，是不能盲目的縮短訊號間距離的，解決的方法只有從編碼著手：

改良的同步記錄法：MFM

MFM（Modified Frequency Modulation）是最早的倍密度記錄法，前一節已提到，FM的缺點是同步訊號佔用了一半的磁片空間，所以MFM只有對於連續的0訊號間才加入同步脈衝來加以隔離。如此一來空間便可增加一倍了（因爲FM的瓶頸是在於頻率最高的1訊號及其前後的同步訊號，而不在於0訊號）。可是資料與同步脈衝是如何區別的呢？答案是：同步訊號第一次是發生在間距爲1.5倍的時候。



圖A－4 各種方式的比較

MFM的困難仍舊是由於訊號的磁力作用及速度變化而使得讀取不易。

再改良的同步記錄法：M FM

M^2FM (Modified Modified Frequency Modulation) 是將 0 訊號間的同步脈衝再予減少，其規則是：只有在連續兩個 0 訊號之後的 0 訊號間才插入同步脈衝。

```
資  料   1 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 1
MFM      1 1   1 1 1   1 1  1 1 1 1 1 1  1   1
M^2 FM   1 1     1 1   1 1    1   1    1 1   1
```

圖A－5　M^2FM與MFM的比較

我們可以看出M^2FM的密度又比 MFM 少了許多。

GCR (Group Code Recording)

無論FM、MFM或M^2FM均是爲了避免'連續的 0 無法分辨'這個問題才加入同步脈衝的，如果能夠將原始位元組編碼成沒有連續 0 位元的狀況，然後將這種經過編碼的位元組寫入磁片，那麼同步脈衝就完全不需要了，這種編碼記錄的方法稱爲 GCR (Group Code Recording)。

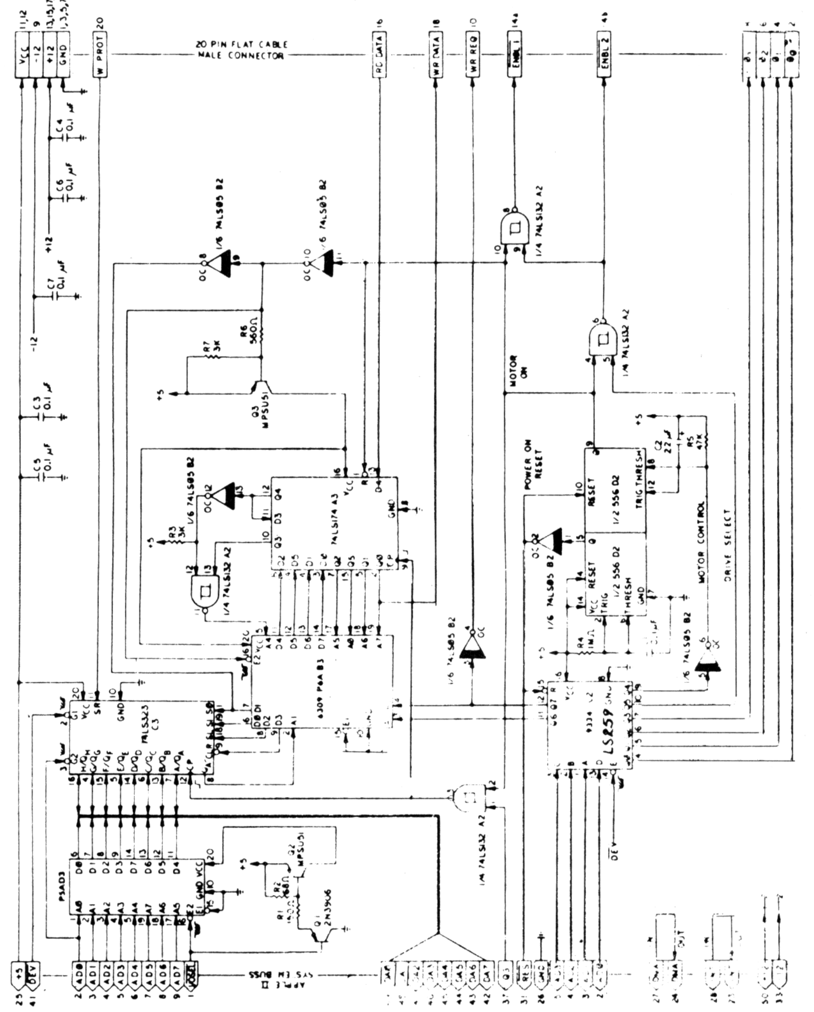
GCR 的編碼方式隨機種而有所不同，常見的是將 4 個位元編成 5 個位元，如下表所示：

原始資料	編碼後
0000	11001
0001	11011
0010	10010
0011	10011
0100	11101
0101	10101
0110	10110
0111	10111
1000	11010
1001	01001
1010	01010
1011	01011
1100	11110
1101	01101
1110	01110
1111	01111

表右邊的編碼也出現連續兩個0的情況，這是因爲一般磁碟機對於兩個以內的0還能正確處理，所以是允許的。依照上表的編碼方式，像' 00011000 '這個位元組，要先分成半位元組' 0001 '與' 1000 '，然後編碼成'11011 '及' 11010 '才寫入磁片。

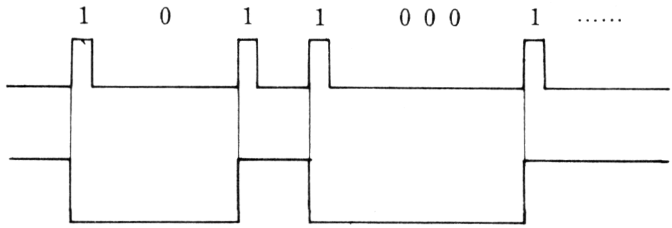
APPLE 的磁片便是以 GCR 的方式來存資料的，不過其編碼方式不同於上表。APPLE 是將6個位元的資料編成8個位元的碼，並且除了不能有連續兩個以上的0之外，還規定最高位元必須爲1，這種 GCR 的編碼方式稱爲6／2編碼法，詳見第6章。

A－2 APPLE讀寫的硬體動作



圖A－6　APPLE的磁碟機介面卡

當 DOS 的RWTS將資料編成6／2 GCR 碼之後，便觸動Q6、Q7將資料由資料線（Data bus）送往介面卡的LS323，將並列的訊號轉換成串行形式。串行位元由LS323 的第8脚 Q_A 送往由6309 及LS174 構成的時序控制器，每次LS 323 輸出的位元值爲′1′時，LS174 的輸出（第2脚Q_0）就反轉，而位元值爲′0′時，則不反轉。其情形如下圖：



這正好是NRZ 的情形！其實無論是FM、MFM、GCR、……所有的磁片記錄法其根本還是NRZ， 只不過 FM 加入 clock 而GCR 使用了編碼而已。LS174 之輸出除了做 NRZ 之外還將每個位元的間隔調成 4 μS， 這便是APPLE 一個位元組要花 $4 \times 8 = 32\ \mu$S來讀寫的原因。

至於資料的讀入，APPLE 使用一個專爲磁碟機而設計的IC MC3470（置於磁碟機內）來將磁片訊號讀取並且整形才送至介面卡，訊號由介面卡的 16 脚（RD DATA）送往LS174 的 13 脚（D_4），LS174 與6309 ROM再度構成一個時序控制器，將訊號處理並送往LS323 以轉換成並列位元組，最後由資料線送往主機，當然這時還是 GCR 碼，必須由第6章所介紹的 POSTNIBL 來加以解碼才能還原爲原來資料。

＊加入 clock 其實也是一種編碼，譬如位置欄編碼便等於是加入 clock 一樣。

B 磁區讀寫程式的操作法

ZAP
NIBBLE AWAY II
COPY II PLUS 4.0
的 SECTOR EDITOR 操作法

本章將幾種常用的磁區修改（SECTOR EDITOR）程式使用法，加以說明，以便本書各章節使用 *。

B－O COPY II PLUS 4.0

啓動COPY II PLUS 4.0程式之後，畫面會出現各種功能的選擇，利用 [←] [→] 鍵來移動反白游標至 [SECTOR EDITOR] 位置，[RETURN] 之後，便出現了如下的畫面：

---

* 請選出您合用的一種（或已有的程式）閱讀即可。

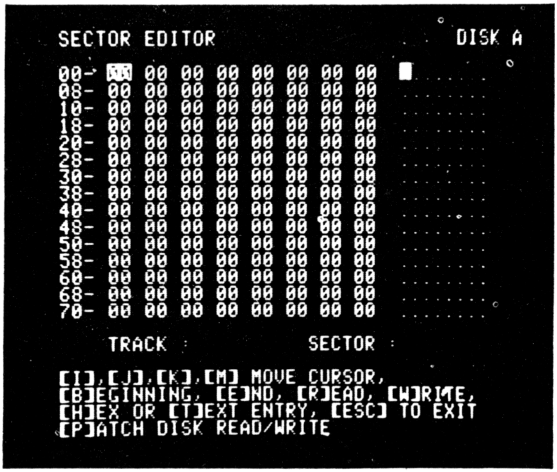


圖 B－0　進入磁區修改狀態

取出COPY Ⅱ PLUS 4.0 磁片，換入實驗磁片（或任何你要做磁區修改的磁片），然後按下 R 鍵，APPLE 便會問你：READ FROM TRACK：？ SECTOR：？ ，如下圖所示

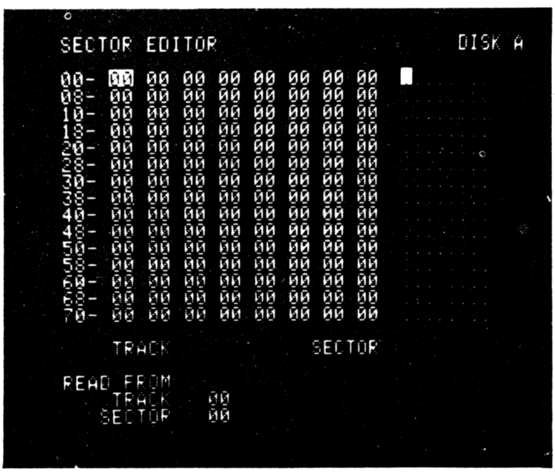


圖 B－1　選擇讀取磁區狀態

譬如說，你想讀取第 00 磁軌的第 00 磁區，則分別鍵入00 [RETURN]，00 [RETURN]，現在磁片上的（$00，$00）磁區內容便被讀入主機的記憶體內（BUFFER），並且在螢幕上顯示了128位元組的資料，這時螢幕右方顯示的是資料的ASCII字符。你可以移動 [I]↑，[J]←，[K]→，[M]↓等鍵來移動反白游標，如果要修改資料，首先要將游標移動到所要修改的位元組位置上，然後鍵入[H]，這時便可以 16 進位數來修改資料了。如果鍵入[T]而不是[H]，則可以直接以 ASCII 字符來修改（對文字檔有效）其內容，按下[　　]鍵可以將反白游標移至下一位置，以便繼續進行修改。修改完畢之後，按下[W]鍵便可以將修改後的資料寫回磁片了。



圖 B－2　讀取磁區資料及修改的情形

B－1 NIBBLE AWAY Ⅱ

NIBBLE AWAY 也是一個拷貝專用的程式，不過它也提供了一個與COPY Ⅱ PLUS類似的磁軌／磁區修改程式（TRACK／SECTOR EDITOR）。啓動NIBBLE AWAY之後，按下[S]便進入了TRACK／SECTOR EDITOR（簡稱 TSE）。TSE 的操作法如下：

[I]、[J]、[K]、[M]：將游標做上、J、右、下移動。

[←]、[→]：將游標做左、右移動。

[A]：將資料以ASCII 字符顯示，若無對應之 ASCII 值，則以反白 16 進位值顯示。

[R]：將所選擇的磁區讀入所選擇的暫存區（BUFFER）。

[W]：將暫存區的值寫入所選擇的磁區。

[T]：改變磁軌號碼。

[S]：改變磁區號碼。

[O]：進入選擇狀態，其操作法如下

- [S]：每按一次，SLOT 號碼加一。
- [D]：使磁碟機（DRIVER）號碼在 1 與 2 之間變化。
- [F]：選擇 13 磁區／16 磁區的格式。
- [C]：改變 checksum 旗號（以決定是否要使用 checksum），如此可以方便讀取非標準型式的磁區結尾記號（closing marker）。
- [RETURN]：離開選擇狀態。

[L]：將磁區資料以 6502 的組合語言方式印在螢幕上，如下例

所示

```
                 SECTOR EDITOR
----------------------------------------------
TRK=00          DISASSEMBLE             SEC=01
----------------------------------------------
   0000-    8E E9 B7      STX    $B7E9
   0003-    8E F7 B7      STX    $B7F7
   0006-    A9 01         LDA    #$01
   0008-    8D F8 B7      STA    $B7F8
   000B-    8D EA B7      STA    $B7EA
   000E-    AD E0 B7      LDA    $B7E0
   0011-    8D E1 B7      STA    $B7E1
   0014-    A9 02         LDA    #$02
   0016-    8D EC B7      STA    $B7EC
   0019-    A9 04         LDA    #$04
   001B-    8D ED B7      STA    $B7ED
   001E-    AC E7 B7      LDY    $B7E7
   0021-    88            DEY
   0022-    8C F1 B7      STY    $B7F1
   0025-    A9 01         LDA    #$01
   0027-    8D F4 B7      STA    $B7F4
```

圖B－3　磁區資料的組合語言列表

使用 [←] 及 [→] 鍵可以令組合語言列表在螢幕上、下捲動。

[　　]：進入資料修改狀態，資料修改的操作如下：

[0]～[9]、[A]～[F]：輸入16 進位值至修改窗內，修改窗的位置是在螢幕上 MODIFY 這個字後面的括弧內。

[　　]：按下空格鍵後，會將修改窗內的值移入游標位置的位元組內，並將游標向右移一格。

[←]、[→]：將游標向前、後移一格。

RETURN ：離開資料修改狀態。

＋ ：將磁軌號碼加一。

－ ：將磁軌號碼減一。

Q ：離開 TSE。

```
                 SECTOR EDITOR
--------------------------------------------------
TRK=00                                      SEC=01
--------------------------------------------------
00- 8EE9B78E F7B7A901 8DF8B78D EAB7ADE0
10- B78DE1B7 A9028DEC B7A9048D EDB7ACE7
20- B7888CF1 B7A9018D F4B78A4A 4A4A4AAA
30- A9009DF8 049D7804 2093B7A2 FF9A8EEB
40- B74CC8BF 2089FE4C 849DADE7 B738EDF1
50- B78DE1B7 ADE7B78D F1B7CEF1 B7A9028D
50- ECB7A904 8DEDB7A9 028DF4B7 2093B7AD
70- E7B78DFE B6186909 8DF1B7A9 0A8DE1B7
80- 38E9018D FFB68DED B72093B7 60000000
90- 000000AD E5B7ACE4 B720B5B7 ACEDB788
A0- 1007A00F EAEACEEC B78CEDB7 CEF1B7CE
B0- E1B7D0DF 60087820 00BDB003 28186028
C0- 3860ADBC B58DF1B7 A9008DF0 B7ADF9B5
D0- 49FF8DEB B760A900 A89142C8 D0FB6000
E0- 1B020A1B E8B700B6 016002FE 0001FBB7
F0- 00B70000 0202FE60 02000000 01EFD800
```

圖 B－4 磁區資料的讀取情形

```
                SECTOR EDITOR
--------------------------------------------------
TRK=01                                       C=08
--------------------------------------------------
00-[U]N V E  R I F Y  @ ! 0    0 ! 0
10- 0   0    0   0    0   @ "  F   4 "
20- F " D #  8 " 0 0  0 @ 0 @  @ @ @ H
30- @ H @ D  @ @ 0 @  @ ! 9    1   1
40- 0 V D S  L R B A  C I O @    P H D
50- B A @    P @ @ ~  @ A @ B  @ A @ G
60- @ A @ ~  ? @ @ ~  ? @ @ ~  ? @ @ ~
70- ~ M G M  L A N G  U A G E    N O T
80-   A V A  I L A B  L E R A  N G E
90- E R R O  R W R I  T E   P  R O T E
A0- C T E D  E N D    O F   D  A T A F
B0- I L E    N O T    F O U N  D V O L
C0- U M E    M I S M  A T C H  I / O
D0- E R R O  R D I S  K   F U  L L F I
E0- L E   L  O C K E  D S Y N  T A X
F0- E R R O  R N O    B U F F  E R S
```

圖 B－5　以 ASCII 字符來顯示磁區資料的情形

B－2 BAG OF TRICKS的ZAP

這是本書所選用的程式，其優點是整個磁區的256個位元組，可以一次顯示在一個畫面上，並且資料值與 ASCII 字符也一併顯示，其缺點是資料顯示時，每一位元組之間沒有空格，不易算出位元組位置來。ZAP 的功能很多，指令有五十幾個，此處只介紹本書將使用的幾種：

Rtt,ss：讀取第 tt 磁軌的第 ss 磁區。整個命令必須連續鍵入，不能有空格。如果有檔案被 OPEN，則此命令是讀入第 tt 錄（Record）的第 ss 位元組。

16進位數字 ：移動游標至此位元組上面，譬如：鍵入 3 B RETURN ，游標就移動到$3B的位置，再鍵入 0 RETURN ，則游標又回到左上角$00 的位置了。

：數字或字串 ：鍵入 ： 及 16 進位數（最多可以 16 位元組），可以將冒號之後的數值存入目前游標所在位址及其後的對應位址內*。若欲存入字串，則需以在字串前後加上單或雙引號，譬如：'STRING' 可以將 STRING 這個字串存入目前游標所在及其後的 5 個位元組內。

WRITEtt，ss ：將目前暫存區（BUFFER）的內容寫入第$tt 磁軌的第$ss 磁區。因此，我們可以用冒號先將 BUFFER 的內容修改並檢查無誤之後再寫入磁片內，以上說明指沒有檔案被 OPEN的情況，若有檔案被OPEN，則資料寫進的是（錄，位元組）而不是（磁軌，磁區）。

Nn或N ：讀由目前磁區算起第n 個磁區的內容，若 n 不標出，則讀下一磁區。以上說明指沒有檔案被OPEN的情況，如果有檔案被 OPEN，則所讀的是錄而不是磁區。

Pn 或P ：讀出目前磁區算起上n 個磁區資料，若 n 不標出，則讀上一磁區資料。若有檔案 OPEN，則所讀的是錄而不是磁區。

---

* 以類如監督程式 2A：03AE4B可以將 03AE4B存入 2A以下的位址內

L'string'：尋找引號內字串的位置，若找到該字串，則游標置於該字串的第一個字元上，若在本磁區找不到，則至下一磁區，一直到找出符合的字串爲止，若找遍全部磁片，則回到原磁區而停止。如果有檔案被OPEN，則以該檔案範圍爲限

LOCK：ZAP程式Boot時，是在LOCK狀態，WRITE 命令不能將資料寫入磁片。

UNLOCK：UNLOCK 命令可以將LOCK狀態取消。

CPM：告訴 ZAP 程式這是一片 16 磁區格式的CP／M磁片。

PASCAL：告訴 ZAP 程式這是一片 16 磁區格式的PASCAL 磁片。

DOS 16：告訴 ZAP 程式這是一片 16 磁區格式的正常磁片。

DOS 13：告訴 ZAP 程式這是一片 13 磁區格式的早期APPLE 磁片。

OPEN'fn'：打開檔名爲 fn 的檔案。

CLOSE：關閉目前打開的檔案。

RLEN ll：OPEN命令自動設定檔案的錄長度爲256位元組，而RLEN ll之後的R或WRITE 指令則以ll錄長度爲來存取資料。

PRINT：將螢幕上資料以印字機印出來。

S，1或2：選擇一號或二號磁碟機來存取資料。

I：將資料以 6502 組合語言方式來印出來字幕上，I 命令每次印滿一個字幕的資料，若已印完該磁區，則停止印字

並回到＄00 位置。按下 RETURN 可以離開 I 狀態而回到一般 ASCII 字符及 16 進位數狀態。

VTOC ：讀取第＄11 磁軌、第＄00 磁區上的VTOC表。

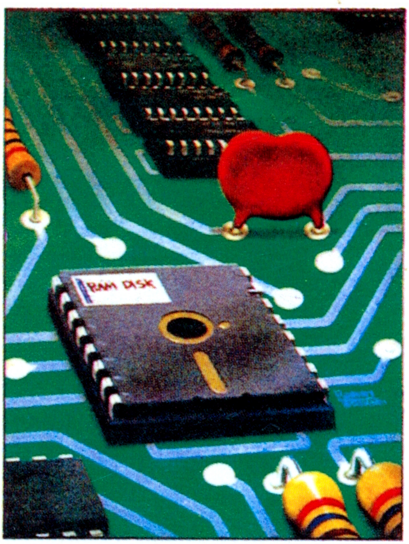
CAT ：讀取磁片上目錄表的頭一個磁區。

STATUS ：顯示 ZAP 目前狀態。

HELP ： HELP 指令可以顯示出 ZAP 命令的使用說明，令您快速的瞭解每一命令的功能。按下 RETURN 可以離開 HELP 而回到原來狀態。

```
00 00110E000000000000000012 ............
0C 0F02C8C5CCCCCFA0A0A0A0A0 ..HELLO
18 A0A0A0A0A0A0A0A0A0A0A0A0
24 A0A0A0A0A0A0A0A002000000         ....
30 000000000000000000000000 ............
3C 000000000000000000000000 ............
48 000000000000000000000000 ............
54 000000000000000000000000 ............
60 000000000000000000000000 ............
6C 000000000000000000000000 ............
78 000000000000000000000000 ............
84 000000000000000000000000 ............
90 000000000000000000000000 ............
9C 000000000000000000000000 ............
A8 000000000000000000000000 ............
B4 000000000000000000000000 ............
C0 000000000000000000000000 ............
CC 000000000000000000000000 ............
D8 000000000000000000000000 ............
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
FC 00000000
```

圖B－6 以ZAP讀取磁區的螢幕


RAM DISK

APPLE II

003

微電腦徹底研究

3

磁碟系統篇

施威銘 著